BATCH VERIFICATION OF DIGITAL SIGNATURES IN IOT

Thesis

Submitted in partial fulfilment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

APURVA S. KITTUR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

SURATHKAL, MANGALORE - 575 025

OCTOBER, 2019

DECLARATION

by the Ph.D. Research Scholar

I hereby declare that the Research Thesis entitled **Batch Verification of Digital Signatures in IoT** which is being submitted to the **National Institute of Technology Karnataka, Surathkal** in partial fulfilment of the requirements for the award of the Degree of **Doctor of Philosophy** in Department of Computer Science and Engineering is a bonafide report of the research work carried out by me. The material contained in this Research Thesis has not been submitted to any University or Institution for the award of any degree.

Apurva S. Kittur, 155009 CS15FV03
Department of Computer Science and Engineering

Place: NITK, Surathkal.
Date: 17 October 2019

CERTIFICATE

This is to certify that the Research Thesis entitled **Batch Verification of Digital Signatures in IoT** submitted by **Apurva S. Kittur** (Register Number: 155009 CS15FV03) as the record of the research work carried out by her, is accepted as the Research Thesis submission in partial fulfilment of the requirements for the award of degree of **Doctor of Philosophy**.

Dr. Alwyn R. Pais
Research Guide

(Signature with Date and Seal)

Chairman - DRPC
(Signature with Date and Seal)

ACKNOWLEDGEMENTS

I take this opportunity to express my sincere gratitude to all the people who encouraged me to complete this dissertation.

First and foremost I would like to thank my supervisor and Head of the Department, Dr. Alwyn R. Pais, for providing me an opportunity to pursue PhD under his guidance. I appreciate his support, patience, motivation, and immense knowledge. I could not have imagined a better mentor for the entire time of research and thesis of my PhD study.

I would also like to thank the research progress assessment committee members, Dr. B. R. Shankar and Dr. Manu Basavaraju, for their insightful comments and encouragement regarding my research work. I also extend my sincere thanks to the entire teaching and non-teaching staff of the Department of Computer Science and Engineering.

I would like to thank the Ministry of Electronics & Information Technology (Meity), Government of India for their support in part of the research. I would also like to thank NITK for providing me the infrastructure and facilities to pursue my research work successfully.

A special thanks to my friends Alok Kumar, Srinivas and Shrutakirthi Godkhindi for being with me all the time as a constant support during the tenure of my PhD. I would also like to thank my friends Nikhil, Ajnas, Tanupriya, Zubair, Siva Kumar Sir, Somesha Sir for all their help during the course of my PhD.

I am deeply indebted to my husband Raghunath Kulkarni, who has been a constant pillar of emotional, technical and motivational support till the end of my PhD. Similarly I am equally thankful to my parents and my loving brother Dr. Amogh, who encouraged me to pursue PhD and my in-laws who have been very supportive towards the course of study.

I thank almighty for providing me the right orientation throughout the path of life and career.

Apurva S. Kittur

ABSTRACT

Internet of Things (IoT) is the interconnectivity of various devices, such as sensor nodes, actuator nodes, gateway nodes, and other devices that have the software, and electronics embedded within them which enables them to exchange data. These devices lack the computation power, memory and battery capacity. The gateway node in IoT handles various responsibilities such authentication, verification, data processing, data encryption, decryption etc. Hence it is important to reduce the bottleneck at the gateway node, so that the network is stable and secure. Therefore security in IoT becomes an important field of research. Digital signatures are one of the ways of authenticating the sender and also to protect the integrity of the data during the communication. Verifying multiple digital signatures together in a batch reduces the computation load and computation time during verification. There are many batch verification schemes designed for popular digital signature algorithms such as RSA, DSS, ECDSA etc. Majority of the batch verification schemes are not lightweight and are prone to attacks.

Even though the contemporary batch verification schemes have evolved with time, but they are not scalable with the increase in batch size. Therefore this research focuses on designing a new batch verification scheme which overcomes the drawbacks and is suitable for IoT. ECDSA digital signature algorithm is a lightweight digital signature algorithm because of its small signature size compared to other schemes for the same level of security. Hence designing a batch verification algorithm for ECDSA* signatures is beneficial in IoT. ECDSA* signature is a modified version of ECDSA signature whose verification time is faster than ECDSA signatures. The proposed batch verification scheme in the research is efficient for verification of multiple ECDSA* signatures and is more secure than the other existing batch verification schemes.

Most of the other existing batch verification schemes do not specify the index of the bad signature. There are many schemes in literature, to identify the bad signature in a given batch, but either they are compute intensive or can not identify all the bad signatures. Hence the research also proposes three bad signature identification schemes

based on hash function and Error Control Codes. After the batch verification test fails, the signatures are verified using proposed schemes to identify the faulty ones. The proposed verification schemes are lightweight compared to sequential verification and other existing verification schemes.

As the aim of research is to implement batch verification in IoT to reduce the bottleneck at the gateway node, the next topic of research is to design a trust model that can decrease the load at gateway node by sharing it. The proposed trust model chooses a set of $Trusted$ nodes from the total available sensor nodes and distributes a set of signatures to each of them. These $Trusted$ sensor nodes verify the signatures using the proposed batch verification scheme. This will significantly reduce the bottleneck at the gateway node without the compromise in security.

Contents

List of Figures vi

List of Tables viii

List of Abbreviations ix

1 Introduction 1

1.1 Batch Verification of Digital Signatures 2

1.2 Digital Signatures: Properties, Applications, and Threats 5

1.2.1 Digital Signature Algorithms 5

1.2.2 Properties 6

1.2.3 Applications 6

1.2.4 Threats on Digital Signatures 7

1.3 IoT and its Security 9

1.4 Motivation 10

1.5 Objectives 11

1.6 Thesis Contribution 13

1.7 Thesis Organization 14

2 Literature Review 15

2.1 Batch Verification Schemes 16

2.1.1 RSA Batch Verification Scheme 17

2.1.2 DSS Batch verification schemes 25

2.1.3 ECDSA Batch Verification schemes 33

2.2 Bad Signature Identification Schemes 47

2.3 Security Trust Models 49

2.4 Research Gaps 52
2.5 Summary 53

3 Batch Verification of ECDSA* Signatures 55
3.1 Definitions and Notations 56
3.1.1 The ECDSA algorithm 58
3.1.2 The ECDSA* algorithm 59
3.2 Proposed Batch Verification Scheme 61
3.3 Security Analysis 65
3.3.1 Possible Attacks on the Existing Schemes 65
3.4 Results and Analysis 68
3.4.1 Verification Times 69
3.4.2 Speedup Values 72
3.4.3 Computation Cost Analysis 76
3.4.4 Running Time Analysis 78
3.5 Summary 78

4 Bad Signature Identification in Batch Verification 81
4.1 Introduction 81
4.2 Preliminaries 83
4.3 Hash Based Verification Scheme 88
4.3.1 Comparative Analysis 89
4.3.2 Verification Time for Hash based verification 92
4.4 CRC based bad signature identification scheme 93
4.4.1 Error Detection Codes 93
4.4.2 CRC Verification Algorithm 97
4.4.3 Security Analysis 97
4.4.4 Comparative Analysis 99
4.4.5 Results for CRC based scheme 100
4.5 LDPC based bad signature identification scheme 102
4.5.1 Designing the Parity Check Matrix 103

4.5.2 LDPC Verification Algorithm . . . 106
4.5.3 Security Analysis . . . 108
4.6 Results . . . 109
4.6.1 Results for the proposed batch verification scheme . . . 111
4.7 SUMMARY . . . 113

5 A Trust Model based Batch Verification of Digital Signatures in IoT 115
5.1 Preliminaries . . . 117
5.1.1 IoT network nodes . . . 117
5.2 Proposed Model . . . 118
5.2.1 Parameters for Node selection . . . 120
5.2.2 Implementation of the ECDSA* batch verification algorithm . . . 122
5.3 Node selection based on Physical Parameters . . . 124
5.3.1 *Avail* node selection algorithm . . . 125
5.4 Node selection based on Security Parameters . . . 127
5.4.1 *Trusted* Node Selection Algorithm . . . 129
5.5 Results and Discussion . . . 130
5.5.1 Ideal Condition Results . . . 131
5.5.2 Practical Condition Results . . . 133
5.5.3 Security Analysis . . . 135
5.6 Summary . . . 137

6 Conclusions and Future Work 139

Bibliography 143

Publications 157

List of Figures

1.1 Basic Structure of IoT 9

4.1 Execution Time when 50% signatures are faulty 101
4.2 Execution Time when all signatures are faulty 102
4.3 Graphical Representation 105
4.4 LDPC encoding and decoding at sender and receiver 108

5.1 Trust Model for Digital Signature Verification in IoT 119

List of Tables

2.1 Comparative analysis of Batch verification techniques for RSA 21
2.2 Characteristics of Batch verification techniques in RSA 22
2.3 Comparative analysis of Batch verification techniques for DSS 30
2.4 Characteristics of Batch verification techniques in DSS 31
2.5 Notations followed in ECDSA 34
2.6 Comparative analysis of Batch verification techniques for ECDSA . . . 38
2.7 Characteristics of Batch verification techniques in ECDSA 38
2.8 Comparative study of Batch verification schemes based on Bilinear Pairing 45
2.9 Characteristics of Batch Verification techniques based on Bilinear Pairing 46
2.10 Various Bad Signature Identification Schemes 48
2.11 Trust Models 50

3.1 Notations followed in chapter 56
3.2 Verification Time for the curve ***(P-192)*** for Single Signer (sec) 70
3.3 Verification Time for the curve ***(P-224)*** for Single Signer (sec) 70
3.4 Verification Time for the curve ***(P-256)*** for Single Signer (sec) 71
3.5 Verification Time for the curve ***(P-192)*** for Multiple Signers (sec) 72
3.6 Verification Time for the curve ***(P-224)*** for Multiple Signers (sec) 72
3.7 Verification Time for the curve ***(P-256)*** for Multiple Signers (sec) 73
3.8 Speedup for the curve ***(P-192)*** for Single Signer 73
3.9 Speedup for the curve ***(P-224)*** for Single Signer 74
3.10 Speedup for the curve ***(P-256)*** for Single Signer 74
3.11 Speedup for the curve ***(P-192)*** for Multiple Signers 75
3.12 Speedup for the curve ***(P-224)*** for Multiple Signers 75

3.13 Speedup for the curve ***(P-256)*** for Multiple Signers 76
3.14 Various Operations of batch verification for Single Signer 76
3.15 Various Operations of batch verification for Multiple Signers 77
3.16 Execution Time for expensive operations 77

4.1 Time required for various Verification operations 92
4.2 Time taken for Generation operation (msec) 100
4.3 CRC Encoding and Decoding times 109
4.4 Hash Encoding and Decoding times 110
4.5 LDPC Encoding and Decoding times 110
4.6 Verification Time for Hash-based verification for Single Signers (sec) . . 111
4.7 Verification Time for Hash based verification for Multiple Signers (sec) 112
4.8 Verification Time for CRC based verification for Single Signers (sec) . . 112
4.9 Verification Time for CRC based verification for Multiple Signers (sec) 113
4.10 Verification Time for LDPC based verification for Single Signers (sec) . 113
4.11 Verification Time for LDPC based verification for Multiple Signers (sec) 113

5.1 Current Consumption by different states of the node 121
5.2 Verification time(sec) for a single signer 123
5.3 Verification time(sec) for multiple signers 123
5.4 Ideal Condition (a) Proposed Model Node Selection (b) Random Node Selection 132
5.5 Ideal Condition (a) Physical Parameter based Node Selection (b) Security Parameter based Node Selection 133
5.6 Practical Condition (a) Proposed Model Node Selection (b) Random Node Selection 134
5.7 Practical Condition (a) Physical Parameter based Node Selection (b) Security Parameter based Node Selection 134

LIST OF ABBREVIATIONS

Abbreviations	**Expansion**
CRC	Cyclic Redundancy Check
CPU	Central Processing Unit
DC	Divide-and-Conquer
DLP	Discrete Logarithm Problem
DSS	Digital Signature Scheme
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
FIPS	Federal Information Processing Standard
GCD	Greatest Common Divisor
GT	Generic Tests
IoT	Internet of Things
LDPC	Low Density Parity Check
NIST	National Institute of Standards and Technology
PKI	Public Key Infrastructure
QoS	Quality of Service
RSA	Rivest, Shamir and Adleman
SOAP	Simple Object Access Protocol
WSN	Wireless Sensor Network

Chapter 1

INTRODUCTION

Internet of Things (IoT) is coined in 1999 by Kevin Ashton. IoT can be defined in many ways (Atzori et al. 2010; Gubbi et al. 2013; Zhu et al. 2010). 'Internet' refers to the interconnectivity of devices to create a network, and 'Things' refers to the embedded objects or devices that can connect to the Internet. One way of defining is, 'it is a network of sensors and smart devices which sense the data which is further processed and analysed in an ubiquitous network.' IoT has seen rapid development in recent years because of its 'smartness.' The various applications of IoT include Smart City (Cocchia 2014; Jin et al. 2014), Smart Home (Du et al. 2013; Jie et al. 2013), Smart Health (Amendola et al. 2014), Transport and Logistic applications (Karakostas 2013), Weather monitoring and Forecast (Ram and Gupta 2016) etc. These applications have millions of devices generating large volumes of data.

The rapid development of IoT has increased the curiosity of the attackers. The devices (nodes) in the IoT network have weak security protocols because of their limited computation ability and energy. Hence the nodes of the network are vulnerable to various kinds of attacks. The IoT network has sensors, actuators, controllers, gateway heads, sink, etc. For our reference, we have broadly classified these devices into sensor nodes and gateway nodes. Sensor nodes includes low computation end devices such as sensors, actuators etc., and gateway node includes gateway, sink, cluster head etc. The gateway nodes have better computation power as compared to sensor nodes. But the overall network is not competent to handle complex cryptographic operations. Hence

the security of IoT network is one of the most researched topics. Authentication is one of the important steps in any digital communication. There are many authentication schemes available which verify the identity of the sender.

Signature is a unique way to identify a signer. The authenticity of a person/ organization/ entity is verified through its signature. A signature can be a handwritten one, or it can be a digital one. Digital signatures are used to verify the content of the received message and the signer's identity in digital communication. The exponential growth of Internet technology has led to the growth in usage of digital signatures. Every signer generates a unique signature using his/her private key. The verifier possesses the public key of the signer and verifies the signature using the same. The signature generated by the signer can be verified by everyone who has access to the public key. Digital signatures (Adleman et al. 1977; Blahut 2014) have variety of applications (Chen et al. 2001; Davies 1983; Serret-Avila and Boccon-Gibod 2004). The applications in industries and organizations have established e-Signature standards based on digital signature technology and certified CAs (Certificate Authority)(Vaeth and Walton 2000). Digital signatures are part of the X.509 standard (Housley et al. 2002), which is an International, well-understood, standards-based technology. This standard also helps to prevent forgery and modifications to documents once the signature is generated.

1.1 BATCH VERIFICATION OF DIGITAL SIGNATURES

Most of the digital signature algorithms are based on Public Key Infrastructure (PKI) (Stallings 2006). These algorithms hence have complex cryptographic operations which require higher computation and energy. Some of the popular digital signature algorithms are Rivest, Shamir, Adleman (RSA) (Bellare and Rogaway 1996), Digital Signature Standard (DSS) (Kravitz 1993), and Elliptic Curve Digital Signature Algorithm (ECDSA) (Koblitz 1998). The concept of batch verification is introduced to reduce the verification time and complexity. Batch verification schemes verify multiple signatures together with signatures either signed by single or multiple signers. Fiat (1989) was the first to introduce the concept of batch verification. Later many batch

verification schemes are introduced for RSA, DSS, and ECDSA.

Batch verification algorithm receives a set of digital signatures as input. The output of the algorithm depends on the type of signatures. If there are one or more invalid/bad signatures in the received batch of digital signatures, then the algorithm returns *False*, else *True*. The words invalid/ faulty/ bad/ illegal are used synonymously to refer to the invalid signatures signed by an unauthorised signer or unidentified signer. When all the signatures in the received set of signatures are valid, then the batch of signatures passes the batch verification test. Most of the existing batch verification schemes do not specify the index of the faulty signature/s.

Therefore to identify the faulty signature/s, the verifier has to verify the signatures from the received batch individually. There are various schemes proposed by Pastuszak et al. (2000a); Ren et al. (2015) which aid in determining the faulty signature/s. Some of the schemes perform worse than individual verification if the number of faulty signatures are more in the batch. There are other coding theory based schemes which need the information on the number of invalid signatures in the received batch beforehand to identify the location. But in practical scenario, these schemes are unimplementable, since the number of faulty signatures cannot be known before verification.

Batch verification has various applications due to its low computation load and time at the verifier, with secured communication. If the batch verification validates a batch of signatures as true, then there are no invalid or bad signatures in the batch. If the batch verification returns false, then there are either one or more bad signatures in the batch. Internet based applications such as air traffic control, marketing, hospital management, online banking transactions, etc as mentioned in Ndiaye et al. (2017), require frequent use of digital signatures. Therefore deployment of batch verification in such scenarios helps in faster communication. The advantages of batch verification can be harnessed in IoT, which has computation and energy limitations. Hence batch verification reduces computation time and energy.

Batch verification algorithms are used to verify the signatures signed using the

following three types:

- **Type 1**: Single signer uses his private key (sk) to generate signatures for multiple messages ($m_1, m_2, \ldots, m_t$). The signatures are verified in a batch of t signatures ($s_1, s_2, \ldots, s_t$) at once.
- **Type 2**: Multiple signers use their private keys to sign multiple messages ($m_1, m_2, \ldots, m_t$). Signatures ($s_1, s_2, \ldots, s_t$) are verified in a batch of t signatures using the batch verification algorithm wherein the signatures correspond to n different signers ($2 \leq n \leq t$).
- **Type 3**: The signatures which can not be categorized in Type 1 and 2 can be categorized in this Type.

There are many digital signature algorithms for which batch signature schemes are proposed. One such digital signature algorithm is ECDSA. ECDSA is considered a lightweight digital signature algorithm because of its reduced key size. ECDSA is derived from Elliptic Curve Cryptography (ECC). ECC is introduced by Koblitz (1998) and Miller (1985) independently, which is one of the most secure cryptosystems. ECDSA is a digital signature scheme of ECC which is similar to ElGamal signature schemes. ECDSA is also popular because of its small key size and signature size. As we know, most of the ElGamal signature schemes have the property that the signature verification takes more time than the signature generation. This property holds true for ECDSA scheme too. Hence verifying such signatures together in a batch reduces the overhead of sequential verification. There are many batch verification techniques available for ECDSA signatures which can verify these signatures at a time. Verifying multiple ECDSA signatures in batches (Cheon and Yi 2007; Karati et al. 2012b) makes the verification more applicable in various real-time applications since it reduces the verification time and CPU consumption. Even though the existing techniques as surveyed in Kittur and Pais (2017), try to reduce the computation time, but most of them are not efficient for larger batch sizes. In our study, we are considering ECDSA* signatures for authentication. ECDSA* signatures are the variation of ECDSA signatures (Antipa et al. 2005) which provide 40% more efficiency in verification time

without compromise in security. Apart from the naive batch verification scheme for ECDSA* signatures, there are no batch verification schemes to verify multiple ECDSA* signatures.

1.2 DIGITAL SIGNATURES: PROPERTIES, APPLICATIONS, AND THREATS

In this section, we discuss the properties every digital signature algorithm must satisfy. And we also list the applications whose efficiency is significantly increased by batch verification of digital signatures. We also elaborate on the various possible threats for the verification of digital signatures in batches.

1.2.1 Digital Signature Algorithms

Digital signature (Goh and Jarecki 2003; Stallings 2006) is a way to authenticate the sender/signer of the signature. It verifies the signer by checking whether he/she is a valid signer who he/she claims to be. Digital signatures are usually developed using Public Key Cryptography (PKC) (Diffie and Hellman 1976).
Digital signature algorithm consists of three phases (Katz 2010):

1. **Key Generation phase:** On the input of security parameter k (in Unary), this phase generates both private key (sk) and public key (pk) as outputs.

2. **Signing phase:** For the security parameter k, this phase takes the private key sk and the message m to be signed as inputs, and signs the message and generates the signature s, $s \xleftarrow{sk} m$, for the message as the output.

3. **Verification phase:** For the security parameter k, the input at this phase is the message m with appended signature s and the public key pk. The verifier uses the public key of the signer to verify that the message has been sent by the sender who he/she claims to be and outputs a single bit b. If $b = 1$ then accept the signature or else reject.

Digital signature should satisfy the following requirements:

- The signature generated by the signer should be properly verifiable by the verifier.

- It should not be possible for any third party to forge signer's signature.
- And during disputes between signer and verifier, the third party should be able to resolve the disputes easily.

1.2.2 Properties

Digital signatures are used to verify the following properties:

- **Signature Authenticity -** Verifies that the signature is actually signed by the authorized signer.
- **Signature Integrity -** Verifies that the original signature has not been altered or modified by the unauthorized party.
- **Signature Non-repudiation -** The sender cannot deny signing the authenticated signature.

1.2.3 Applications

Batch verification of digital signatures has a variety of applications (Kinnis and Sit 2005; Naor and Yung 1989). Batch Verification verifies authenticity, integrity and non-repudiation properties of digital signatures for these applications. The various applications where the deployment of batch verification yields better results are:

- **e-cash applications (Claessens et al. 2002; Furnell and Karweni 1999):** This application is an example of Type 1 batch verification signing where multiple e-coins signed by a bank can be verified by consumers/merchants in batches to check the validity of e-coins to achieve a quick transaction.
- **e-Voting system:** This is a typical example of Type 2 batch verification signing where Millions of voters sign the ballot and each ballot has to be validated by verifying the signature of the voter. Ballots are verified in batches to accelerate the counting.
- **WSN and IoT (Suo et al. 2012; Zhang et al. 2014):** WSNs and IoT use Public Key Cryptosystems very widely. When millions of sensor nodes send data

continuously, it is crucial to check the authenticity and integrity of the sending node and data respectively. This application belongs to Type 2 kind of batch verification signing where millions of sensors sign the data and send across, whereas, at the processing side, the data received needs to be verified in batches to get quicker results in a real-time environment.

- **Outsourced Database (Mykletun et al. 2006):** Many clients query signed request messages simultaneously; the server has to authenticate them through Batch verification. It is a typical example of Type 2 batch verification signing, where multiple clients are querying through signed message requests which can be verified in batches at the server for faster results.

- **Intelligent cars (Van Arem et al. 2003; Wang et al. 2006):** Intelligent cars are the recent interest of various governments and industries, where these cars communicate with each other and share the transportation infrastructure to avoid accidents and help to prevent traffic congestion. It is also a Type 2 signing category of batch verification where multiple cars can be authenticated through verification of their signature. Since multiple cars will be communicating simultaneously, it is required to verify them in batches to process the data quickly to make further decisions.

- **Mixnet (Abe 1998; Lee et al. 2003):** It is a network of routing protocols where multiple signed messages from multiple senders are authenticated, shuffled and sent to the next level of proxy servers for authentication. Therefore in Type 2 batch verification, multiple messages are verified in batches by proxy servers at each layer before reaching the destination.

1.2.4 Threats on Digital Signatures

There are many possible attacks on the digital signatures (Boneh et al. 1999; Naor and Yung 1990; Nguyen and Shparlinski 2002; Nguyen and Stern 2001). The attacker may be the signature generator himself, or verifier or it may be a third party. The different possible attacks (Kocher 1996; Stallings 2006; Vaudenay 2003) on the digital signatures are:

- **Key-only attack:** In this attack, the adversary has the information about only the signer's public key. With this available information, he tries to generate the fake message - signature pairs (m_i, s_i).

- **Known message attack (Cheon 2002):** The adversary has access to a certain message - signature pairs (m_i, s_i), where the messages are not chosen by the adversary. And depending on the available information, the adversary tries to learn the signature pattern.

- **Generic chosen-message attack (Pfitzmann and Pfitzmann 1989):** In this attack, the adversary has a batch of messages (m_i) which are independent of the signer's public key pk. Therefore the entire message list is prepared without the knowledge of signatures. Then the adversary gets corresponding valid message - signature pairs (m_i, s_i) for the batch of messages through which he/she tries to learn the signature pattern.

- **Directed chosen-message attack:** In this attack, the adversary chooses a set of messages (m_i) after learning the public key pk of the signer. In this attack also, the list of messages is created without seeing the signatures. Then the adversary gets corresponding valid message - signature pairs (m_i, s_i) for the batch of messages through which he/she tries to learn the signature pattern.

- **Adaptive chosen-message attack (Goldwasser et al. 1988):** Attacker has access to the public key pk of the signer and also he/she has a few valid message - signature pairs (m_i, s_i). Based on this information, the attacker tries to request for additional signatures for the chosen messages by considering the actual signer as 'an oracle' to understand the signature pattern.

All the attacks discussed above are not only applicable to individual verification but also for batch verification. Therefore if the attack successfully bypasses the individual verification test, then there is a high probability that it also bypasses the batch verification test.

1.3 IOT AND ITS SECURITY

Security in IoT is essential as the network of IoT is fast growing. In IoT applications, it is important that the data access and data transfer need to be authenticated and verified. But the nodes in IoT do not have sufficient computation power, memory, and energy. Hence attackers will try to intrude repeatedly. Therefore it is vital to have secure and lightweight authentication schemes in IoT to prevent various attacks.

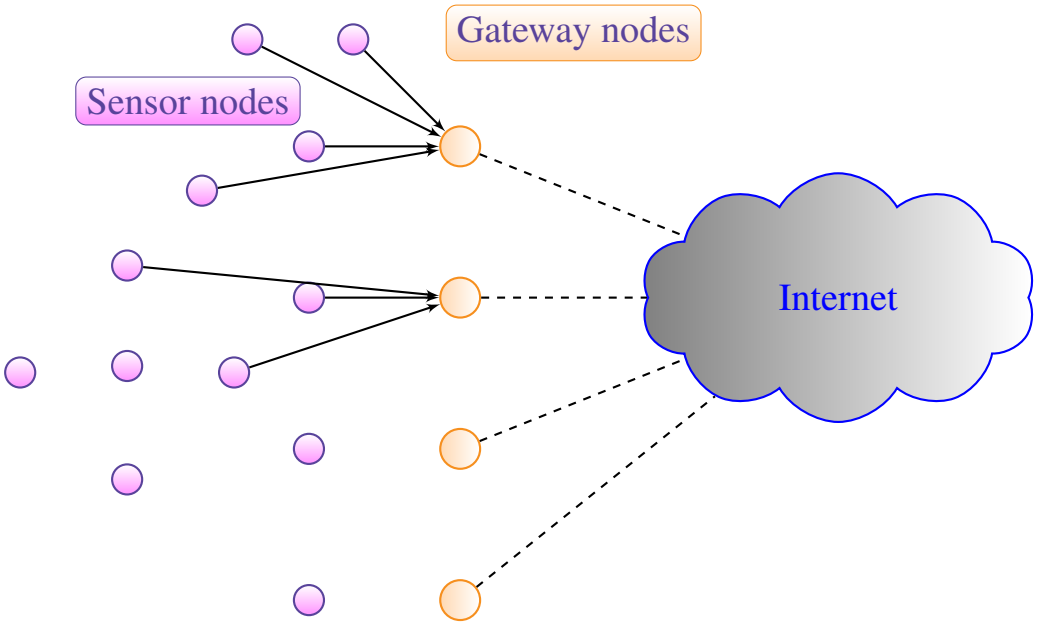

Figure 1.1: Basic Structure of IoT

The gateway nodes in the IoT network act as a bridge between sensors and internet as depicted in Figure 1.1. The Figure 1.1 shows that the sensor nodes, denoted by purple colour send their data regularly to gateway nodes, represented in orange. The gateway node has to handle this huge amount of data traffic coming from sensor nodes before communicating it to outside world through Internet. The gateway nodes collect data from the sensor nodes, verify and normalize the information received for further processing and storage, and they are also responsible for providing security (Kittur et al. 2017). These nodes initially authenticate the sensor node before the exchange of data. Hence they play the role of a firewall by providing the security to sensor nodes as well as to the Internet. Hence the gateway node has heavy computation to do which creates a bottleneck. Therefore, developing techniques and models to reduce this bottleneck is

needed without compromise in security.

Authenticating every data being exchanged in IoT is a challenge. Individual verification of digital signatures reduces the performance of the real-time IoT system. If the signatures are verified in batches, then the verification time can be significantly reduced. Batch verification has two main advantages: one is decreased computation load, and the other is reduced computation time at the verification side without compromise in security.

1.4 MOTIVATION

The Internet of Things (IoT) can be defined as "a pervasive and ubiquitous network which enables monitoring and control of the physical environment by collecting, processing, and analyzing the data generated by sensors or smart objects." Security and privacy are the key issues for IoT applications, and still face some enormous challenges (Jin et al. 2014). These challenges give rise to various security and privacy problems. We should pay more attention to the research issues such as confidentiality, authenticity, and integrity of data in the IoT. Security in IoT is an emerging and significant field of research. There are various security protocols used by IoT devices based on the security requirements of the applications where they are deployed. Because of the limitation of the sensor nodes, it is difficult to implement computation intensive cryptographic algorithms that need more energy and computation power. Hence efficiently implementing secure authentication schemes is an important challenge. Therefore developing secure protocols for IoT nodes and efficiently implementing those protocols without significant reduction in performance is a major area of study. In applications of IoT, the extent of security is inversely proportional to the node's performance.

One way of securing IoT is by implementing the digital signatures in IoT. But digital signatures use PKI, which is compute-intensive. Hence implementation of batch verification of digital signatures is efficient in such an environment. There is an enormous amount of data being exchanged every second in IoT which needs to be verified before processing further. Hence this verification can be achieved through batch

verification. Most of the digital signature algorithms are not lightweight and are not suitable for IoT. ECDSA* digital signature algorithm is considered lightweight because of its smaller key size. Hence it is important to develop a batch verification scheme which verifies multiple ECDSA* signatures to speed up the verification.

Implementing a new batch verification scheme in IoT is not easy considering its limitations. Hence a trust model is necessary, that accounts security and efficiency requirements in IoT environment for implementing the batch verification. Model should consider various parameters of the nodes such as battery life, type of task the node is performing at the given time, and other details. We know that the gateway node performs major processing and verification of the data received from the sensor nodes. Hence the model for implementing the batch verification should also take into account the bottleneck created due to this and should also develop ways to reduce it.

1.5 OBJECTIVES

Though some of the existing batch verification schemes are fast and suitable for IoT applications, they have certain weaknesses. A few of the schemes are not practically implementable, and others are vulnerable to forgery attacks and some are not flexible for varied batch sizes. Also, there are various techniques to identify the location of a bad signature proposed by researchers. Since most of the batch verification schemes do not identify the location of the bad signature, there are separate schemes available to identify the location of the bad signatures. Since we aim to design a batch verification scheme for IoT, it is important that the schemes should be less computational and less time-consuming.

Also efficiently implementing such a batch verification scheme in IoT is important. The scheme should not affect the lifetime of the network significantly. Hence it is important to share the load of verification at the gateway node with other nodes in the network (Frank 2013). But load sharing needs secure and trusted nodes, which can be relied upon without affecting their standard functioning. There are many trust models available in literature which concentrate on choosing the trusted nodes properly. But these models do not consider the node's availability since they are mostly designed for

P2P network, WSN network, where the nodes are designed differently.

Therefore our first objective is to design a batch verification scheme suitable for IoT applications. ECDSA* signatures, an existing modified version of ECDSA signatures, is lightweight signature scheme as well as it reduces the verification time significantly. Hence our aim is to design a batch verification scheme for ECDSA* signatures which can further reduce the verification time. The scheme should be secure against forgery attacks and should be efficient. The scheme should also be efficient for varied batch sizes. A batch verification scheme just alone cannot identify the location of the bad signature/s. Hence it is important to have an efficient bad signature identification scheme which can identify the location of bad signature/s when the batch verification test fails.

Our second objective to design a bad signature/s identification scheme for the batch of signatures that fails the batch verification test. The goal is to develop a scheme that performs better than individual verification in all the cases without compromise in security. The existing scheme based on Hamming Codes has the limitation that, it needs to know the number of bad signatures before verification. Hence such schemes are practically difficult to implement. Thus proposing a new scheme that can overcome such a drawback, is important. Some of the schemes perform worse than individual verification in certain cases. Hence it is important to have low compute-intensive bad signature identification scheme that is suitable for IoT applications.

The third objective of our research is, designing a trust model that can implement the proposed batch verification scheme for IoT environment efficiently and reduce the bottleneck at the gateway node. As we know that the sensor nodes in IoT have low computation power and energy, sharing loads with other nodes helps reduce the bottleneck. But the battery life of sensor nodes is less and are more vulnerable to attacks. Hence a proper model is needed where the gateway can trust the sensor node and distribute the load to it. Therefore our aim is to design a reputation based trust model which can perform efficiently without compromise in the security of the network.

To summarize, the primary objectives of the research are:

- Design, analysis and implementation of a new batch verification scheme for ECDSA* signatures.

- Design a bad signature identification scheme to identify the location of the bad signature in the given batch of ECDSA* signatures suitable for IoT.

- Design a trust model to implement the proposed batch verification scheme in IoT to reduce bottleneck at the gateway node.

1.6 THESIS CONTRIBUTION

IoT constitutes a big heterogeneous network. There are various devices with different capabilities from computation and communication point of view. Due to this heterogeneous property, IoT poses lot of security challenges. We have developed a low compute-intensive approach to implement digital signatures in IoT. The major contributions of the thesis are:

- We provide a detailed study of various batch verification schemes designed for digital signature algorithms (Kittur and Pais 2017). Additionally, we have made a comparative analysis of these schemes based on their characteristic and security features, along with the pros and cons of each scheme.

- We have developed a new batch verification scheme which is faster as well as secure compared to existing schemes (Kittur and Pais 2019a). Our scheme remains efficient regardless of the batch size of the signatures.

- We also have provided a detailed comparison of our scheme with the existing schemes by providing the execution time details of various ECC curves for both single and multiple signers.

- Most of the existing batch verification schemes do not identify the location of the bad or faulty signature. We have designed three faulty signature identification techniques based on the hash function and coding theory to identify the faulty one in a given batch of signatures.

  - First scheme is based on the hash function which is lightweight and secure (Kittur et al. 2019).
  - Second scheme is error detection code based, which efficiently identifies the faulty signatures in a batch based on the Cyclic Redundancy Check (CRC) codes (Kittur et al. 2019).
  - And the third scheme is error correction code based, which identifies the faulty signatures in a batch based on the Low-Density Parity-Check (LDPC) codes.

- A trust model is designed which helps in implementing the batch verification scheme for IoT (Kittur and Pais 2019b). We have developed two algorithms for the model which help in choosing the trusted nodes which perform batch verification.

1.7 THESIS ORGANIZATION

The rest of the thesis is organized as follows: Chapter 2 is the Literature Review of various batch verification schemes available for different digital signature schemes and various trust models suitable for IoT. Chapter 3 explains the new batch verification scheme for ECDSA* signatures. In Chapter 4, we introduce three schemes for identifying the faulty signatures in a given batch. In Chapter 5, a new trust-based model for implementing the batch verification in IoT is proposed. Conclusion and future scope of the work are discussed in Chapter 6.

Chapter 2

LITERATURE REVIEW

IoT network is an interconnection of various heterogeneous networks. These networks consist of various sensor nodes which have low processing power. In such a heterogeneous network, security of the devices plays an important role. The sensor nodes in the network are vulnerable to attacks because of the lack of standard security protocol. The most secure protocols need heavy computations and energy, which the sensor nodes cannot support. Authentication is the important part of security in all the networks. As discussed in Chapter 1, the advantage of batch verification can prove to be efficient for IoT applications. Therefore to design a batch verification scheme for IoT network, it is important to study various existing batch verification schemes. Most of the existing batch verification schemes do not identify the index of the faulty signature in the batch, that fails the batch verification test. The other existing schemes to identify the location of the bad signature have various limitations, which make them difficult to implement in practical conditions.

As discussed in Chapter 1, gateway node plays an important role in IoT network as it handles many responsibilities. Therefore to reduce the load at the gateway node, our idea to share the load with other sensor nodes requires choosing trusted nodes carefully. There are various trust models in the literature, designed to carefully choose trusted entities implemented in various networks using different parameters.

In this chapter, we survey many batch verification schemes designed to speed up the verification process of various digital signature algorithms. The different digital

signatures considered for the study are RSA, DSS, and ECDSA. There are multiple batch verification schemes for verifying each of the digital signature algorithms. We have made a comparative study of these batch verification schemes, to understand the properties of these schemes along with their pros and cons. We have even classified them based on whether the schemes identifies the faulty signature along with verification or just verifies the batch of signatures.

From the survey of the batch verification schemes, it is evident that most of the batch verification schemes do not identify the bad signature in the batch. There are schemes which do identify the bad signature/s, but they suffer from various disadvantages. Therefore, we have surveyed these existing bad signature identification schemes and compared their efficiency to analyse whether they perform well in IoT applications. Hence once the authentication schemes are analysed and compared, the next step is implementation in IoT, so that the load on the gateway node is reduced and the security is improved without significant hit on the performance.

A trust model that can reduce the bottleneck at the gateway node by sharing the load to other nodes. The model should be designed such a way that it can improve the performance at the gateway node as well as maintain the security of the network. There are multiple trust models in the literature designed for various networks to evaluate the trust value based on different parameters. These models differ in the parameters used for evaluating the trust of the entity.

We start our survey with various batch verification schemes for RSA, DSS, and ECDSA. Then we survey various bad signature identification schemes. We also survey various trust models for various networks and compare them. This will help us design an efficient trust model for IoT.

2.1 BATCH VERIFICATION SCHEMES

In this section, we list various batch verification schemes to verify multiple digital signatures. The schemes are categorized based on the digital signature for which they are designed. In our study, we have categorized them based on whether the batch verification scheme is designed for RSA, DSS, and ECDSA.

2.1.1 RSA Batch Verification Scheme

RSA public key Algorithm (Bellare and Rogaway 1996; Rivest et al. 1978) was developed by Ron Rivest, Adi Shamir, and Leonard Adleman in 1977 (Adleman et al. 1977). Soon it became popular and was adopted by many standard bodies. It is based on the difficulty of finding the factors for the product of two large prime numbers.

(a) RSA Digital Signature Algorithm

The RSA digital signature algorithm is based on the RSA encryption scheme. Hence understanding the steps of signature generation and verification is important.

The steps of RSA digital signature are as follows:

1. **Key Generation Phase**
   - Select two large distinct random primes *p* and *q* of bit-length $\frac{l}{2}$, where l is the security parameter.
   - Compute $n = p * q$.
   - Compute the totient value: $\varphi = (p-1)(q-1)$.
   - Select random integer e with $1 < e < \varphi$ and gcd$(e, \varphi) = 1$.
   - Using extended gcd algorithm, compute unique integer *d*, $1 < d < \varphi$, such that $ed \equiv 1 \mod \varphi$.
   - The public key is (n, e) and private key is (d).
2. **Signing**
   - Calculate the hash value for the message m, using cryptographic hash function $M = H(m)$,
   - Sign the hash value with the private key, compute signature $s = M^d$ mod *n*.
   - Signature s is appended to the message and sent across.
3. **Verifying**
   - After receiving the signature s', the verifier calculates $h' = (s')^e$ mod n, as the verifier has public key of the signer for verification.

- Then the verifier also calculates the hash value of the received message $M = H(m)$
- If $M \stackrel{?}{=} h'$, then $Accept$ the signature, else $Reject$.

There are different batch verification techniques introduced to verify RSA signatures in batches. These techniques are divided into two categories. In the first category, the techniques verify the existence of a bad signature in a given batch of digital signatures. And in the second category, the techniques also identify the location along with the existence of bad signatures.

(b) Identifying Existence of Bad-Signature

In this subsection, we discuss the existing techniques for batch verification of RSA digital signatures which identify the existence of bad signature in a given batch of digital signatures. Harn (1998b), Min-Shiang et al. (2001) and Bao et al. (2006) have proposed different techniques for identifying the bad signature in a batch of digital signatures.

(a) In the technique proposed by Harn (1998b), the message to be sent is first hashed, then signed and the signature generated is appended with the original message and sent to the verifier. The equation proposed for signature verification at the verifier is given in Equation 2.1,

$$(\prod_{i=1}^{t} s_i)^e \stackrel{?}{=} \prod_{i=1}^{t} H(m_i) \bmod n \tag{2.1}$$

From the Equation 2.1, it is clear that, after receiving the signatures $s_1, s_2, \ldots s_t$, multiply all the s_i values and create an exponent of the product to the power e. Then all the hash values of the signatures are multiplied independently and the product of both the expressions are compared and verified. If the values match, then there is no bad signature in the batch, else there is an existence of one or more bad signatures. This technique is efficient for Type 1 way of signing signatures for batch verification where a single signer signs multiple signatures. The number of modular exponentiations required at the verification side is one and the number of modular multiplications required is $2t - 2$. The advantage of this

scheme is that it reduces the number of modular exponentiation operations to verify a batch of t signatures compared to the individual verification. The disadvantage of the technique is, it is not secure against adaptive chosen message attack by the signer, and it does not take into account the concept of multiple signers.

The work of Hwang et al. (2000) mainly concentrates on the Harn (1998b) scheme of batch verification of RSA signatures where weaknesses of the scheme are discussed. The possible adaptive chosen message attack by the signer during batch verification using Harn (1998b)'s scheme has been discussed. In the first attack, suppose 3 signatures have to be signed, then the sender signs $s'_1 = H(m_2)^d \mod n$, $s'_2 = H(m_3)^d \mod n$, $s'_3 = H(m_1)^d \mod n$, then sends (m_1, s'_1), (m_2, s'_2), (m_3, s'_3) to the verifier. The verifier multiplies all $s'_i \mod n$ and all $H(m_i)^d \mod n$ separately and compares the two values to verify the sender as legitimate. Now the sender may deny by proving $H(m_i) \neq (s'_i)^e \mod n$. The other attack by the signer is that, the sender sends r signatures $s'_i = a_i * s_i,\ i = 1, 2, \ldots, r$, where a_i is the number that satisfies $a_1 * a_2 * \cdots * a_r = 1$, and s'_i is the forged signature of s_i generated by signer so that the forged signatures pass the batch verification test.

(b) In the technique proposed by Min-Shiang et al. (2001), which is suitable for Type 1 batch verification signing where single user signs multiple messages, the equation for verification at the verifier is given in Equation 2.2,

$$(\prod_{i=1}^{t} s_i^{v_i})^e \stackrel{?}{=} \prod_{i=1}^{t} H(m_i)^{v_i} \mod n \tag{2.2}$$

where v_i is a small random number generated by the verifier, which is used as an exponent for verification of the signatures. This results in $2t + 1$ number of modular exponentiations and $2t - 2$ number of modular multiplications. Therefore if the attacker wants to forge the signature, then he has to guess the value of random number v_i, such that the value has to satisfy the equation, $\prod_{i=1}^{t} s_i^{v_i} = \prod_{i=1}^{t} (s'_i)^{v_i}$, where s'_i is the forged signature. Therefore it is difficult for the attacker to forge the signature

in this technique. The main point of concern in this technique is definitely the increased number of exponentiation operations due to inclusion of new integer.

(c) The scheme proposed by Bao et al. (2006) makes sure that the signatures are generated only with the valid private key. This scheme is efficient for Type 1 batch verification signing. The verifier makes slight modification to the Min-Shiang et al. (2001) scheme and is given by Equation 2.3,

$$(\prod_{i=1}^{t} s_i^{v_i})^{2e} \stackrel{?}{=} \prod_{i=1}^{t} H(m_i)^{2v_i} \bmod n \tag{2.3}$$

where v_i is the random number generated by the verifier. This scheme requires $2t + 1$ number of modular exponentiations, $2t$ number of modular multiplications for batch verification at the verifier. This method is secure against the adaptive chosen message attacks by the signer where he denies signing message. But there is one more possibility of attack with probability $1/a$, where $a \geq 2$, if the signer chooses w such that it satisfies $w^a \equiv 1 \mod n$ in $s_i' = s_i * w$. And to avoid such a possibility, the Equation 2.3 is modified as follows,

$$(\prod_{i=1}^{t} s_i^{v_i})^{be} \stackrel{?}{=} \prod_{i=1}^{t} H(m_i)^{bv_i} \bmod n, \tag{2.4}$$

where $b = \prod_{j=2}^{a} j$. This adds little exponentiation, but provides more security. Therefore Equation 2.4 is more secure than Equation 2.3.

Table 2.1 compares the various batch verification schemes for verifying RSA digital signatures. The various parameters considered for comparison of batch verification schemes are:

- Possibility of Forging: This parameter indicates whether the scheme is vulnerable to forgery attacks.
- Forger: This parameter indicates who can forge the signatures if there is a possibility of forgery.
- No. of Modular Exponentiations: This parameter provides the runtime analysis of the schemes. The variable t is the batch size.

Table 2.1: Comparative analysis of Batch verification techniques for RSA

Technique	**Possibility of Forging**	**Forger**	**No. of Modular Exponentiations**	**Index of bad signature**	**Pros**	**Cons**
Harn (1998b)	yes	signer	constant	no	Reduces the computation load and time at verifier.	There is possibility of Adaptive-chosen message attacks by sender.
Min-Shiang et al. (2001)	yes	signer	linear function of t	no	Provides security against Adaptive chosen-message attacks by signer by 50%.	Increases the computation load at verifier.
Bao et al. (2006)	yes	signer	linear function of t	no	Improves security of the technique Min-Shiang et al. (2001) to reduce the possibility of chosen-message attacks.	Cannot eliminate the possibility of Adaptive chosen message attack by signer completely.
Changchien et al. (2002)	yes	signer	constant	yes	Identifies the location of bad signature accurately.	Increases the number of modular multiplication operations and carries same disadvantages as of Harn (1998b) scheme.
Li et al. (2010)	no	-	linear function of t	yes	Identifies faulty signature location without individual verification.	Exponentiations increase as the number of bad signatures increases.
Ren et al. (2015)	no	-	linear function of t	yes	Performs better irrespective of the number of bad signatures in the given batch.	Better performance than Li et al. (2010) scheme but not as good as Harn (1998b) scheme.
Seungwon et al. (2006)	no	-	linear function of t	yes	Efficient for identifying bad signature position.	Increased number of modular multiplications reduces performance of the verification.

- Index of bad signature: This parameter indicates whether the scheme identifies the location of the faulty signature.
- Pros & Cons: These columns indicate the strengths and weaknesses of the schemes.

Table 2.2 presents the comparison of the schemes based on the security parameters. Also, the column **Computation Overhead** indicates whether the scheme is computation expensive or not. If the number of modular

Table 2.2: Characteristics of Batch verification techniques in RSA

Technique	Authenticity	Integrity	Non-repudiation	Computation Overhead
Harn (1998b)	✓	-	-	low
Min-Shiang et al. (2001)	-	✓	-	high
Bao et al. (2006)	✓	✓	-	high
Changchien et al. (2002)	✓	-	-	low
Li et al. (2010)	✓	✓	✓	high
Ren et al. (2015)	✓	✓	✓	high
Seungwon et al. (2006)	✓	✓	-	high

exponentiations is high, the scheme requires more computation.

(c) Identifying location of bad signature

In this subcategory we discuss various schemes proposed for batch verification of RSA digital signatures which identify the location of the bad signature/s in a given batch of digital signatures.

(a) Changchien et al. (2002) proposed an extension to Harn (1998b) scheme. This technique is applied to a batch of signatures, when there is a bad signature in a batch identified by Harn (1998b) scheme. This scheme belongs to Type 1 batch verification signing, since it is not efficient in case of multiple signers. To locate the index of bad signature, hash function is redefined so that the value of one-way hash function $H(.)$ is prime and $\prod_{i=1}^{t} H(m_i) \leq n$, and let the length of $H(.)$ is $\lfloor \frac{|n|}{t} \rfloor$ bits. The verifier calculates $L = (\prod_{i=1}^{t} s_i)^e$ and verifies if $L \mod H(m_i) = 0,\ i = 1, 2, \ldots, t$. The verifier has to verify L with the hash values of the messages individually by the modular operation. The scheme requires one modular exponentiation, $t - 1$ modular multiplications, and t modulo operations. This scheme increases the number of modulus operations. And also the scheme fails to identify the location of the bad signature, if there are more than one bad signatures in the batch.

(b) Li et al. (2010) proposed a batch verification scheme for a batch of t

signatures signed according to Type 1 batch verification signing. The verifier generates an $M \times N$ matrix where $M * N \geq t$ where t is the batch size and also generates t random numbers r_i, $i = 1, 2, \ldots, t$ and $r_i \in \{1, 2, \ldots, t\}$. The elements of the matrix are the signatures arranged as follows:

$$s(M,N) = \begin{cases} s(\lceil \frac{r_i}{N} \rceil, N), & \text{if } r_i \bmod N = 0 \\ s(\lceil \frac{r_i}{N} \rceil, r_i \bmod N), & \text{otherwise} \end{cases}$$

After the assignment, the verifier verifies the rows and columns using Harn (1998b) scheme. All the rows and columns are independently verified through Equations 2.5, 2.6, 2.7 and 2.8.

At the row side:

For the first row,

$$\left(\prod_{i=1}^{N} s_{(1,i)} \right)^e \stackrel{?}{=} \prod_{i=1}^{N} H(m_{(1,i)}) \bmod n \tag{2.5}$$

For M^{th} row,

$$\left(\prod_{i=1}^{N} s_{(M,i)} \right)^e \stackrel{?}{=} \prod_{i=1}^{N} H(m_{(M,i)}) \bmod n \tag{2.6}$$

At the column side:

For the first column,

$$\left(\prod_{i=1}^{M} s_{(i,1)} \right)^e \stackrel{?}{=} \prod_{i=1}^{M} H(m_{(i,1)}) \bmod n \tag{2.7}$$

For the N^{th} column,

$$\left(\prod_{i=1}^{M} s_{(i,N)} \right)^e \stackrel{?}{=} \prod_{i=1}^{M} H(m_{(i,N)}) \bmod n \tag{2.8}$$

Whenever there is an occurrence of bad signature, it will be detected in both the row and column verification. Depending on the location of bad signature, the corresponding row and column fail the Harn (1998b)'s test and the location of the signature is found. During the batch verification at the verifier, the number of modular exponentiations is $M + N$ and the number of modular multiplications is $2(2MN - M - N)$. This scheme is efficient for a single occurrence of a bad signature. If the number of bad signatures in a given batch increases then efficiency decreases. The adaptive chosen message attacks can become a challenge to this scheme too.

(c) Ren et al. (2015) came up with the technique where a cubical arrangement of signatures is considered for identifying the location of the bad signature. Like the previous technique by Li et al. (2010), this also is efficient for Type 1 way of signing for batch verification. For a given t number of signatures, identify the smallest number 'M' such that the dimension $M^3 \geq t$. Next step is to identify t random numbers r_i, where $r_i \in \{i = 0, 1, \ldots, M^3 - 1\}$ and $i = 1, 2, \ldots, t$. The matrix or the cube is $M \times M \times M$ and the signatures are placed according to $r_i = xM^2 + yM + z$ and $x, y, z \in \{0, 1, \ldots, M-1\}$. For the plane of x-axis the verification Equations are 2.9 and 2.10:

for $x = 0$,

$$\left(\prod_{i=0}^{M-1}\prod_{j=0}^{M-1} s_{(0,i,j)}\right)^e \stackrel{?}{=} \prod_{i=0}^{M-1}\prod_{j=0}^{M-1} H(m_{(0,i,j)}) \tag{2.9}$$

for $x = M - 1$,

$$\left(\prod_{i=0}^{M-1}\prod_{j=0}^{M-1} s_{(M-1,i,j)}\right)^e \stackrel{?}{=} \prod_{i=0}^{M-1}\prod_{j=0}^{M-1} H(m_{(M-1,i,j)}) \tag{2.10}$$

For the plane of y-axis the verification Equations are 2.11 and 2.12:

for $y = 0$,

$$\left(\prod_{i=0}^{M-1}\prod_{j=0}^{M-1} s_{(i,0,j)}\right)^e \stackrel{?}{=} \prod_{i=0}^{M-1}\prod_{j=0}^{M-1} H(m_{(i,0,j)}) \tag{2.11}$$

for $y = M - 1$,

$$\left(\prod_{i=0}^{M-1}\prod_{j=0}^{M-1} s_{(i,M-1,j)}\right)^e \stackrel{?}{=} \prod_{i=0}^{M-1}\prod_{j=0}^{M-1} H(m_{(i,M-1,j)}) \tag{2.12}$$

For the plane of z-axis the verification Equations are 2.13 and 2.14:

for $z = 0$,

$$\left(\prod_{i=0}^{M-1}\prod_{j=0}^{M-1} s_{(i,j,0)}\right)^e \stackrel{?}{=} \prod_{i=0}^{M-1}\prod_{j=0}^{M-1} H(m_{(i,j,0)}) \tag{2.13}$$

for $z = M - 1$,

$$\left(\prod_{i=0}^{M-1}\prod_{j=0}^{M-1} s_{(i,j,M-1)}\right)^e \stackrel{?}{=} \prod_{i=0}^{M-1}\prod_{j=0}^{M-1} H(m_{(i,j,M-1)}) \tag{2.14}$$

Therefore it becomes easy to identify bad signature in a batch by checking the intersection of all the axes and identifying the location where the verifications fail. Number of modular exponentiations is $3M$ and

the number of modular multiplications is $3M^3$. Here the number of exponentiation computations does not abnormally increase as the number of faulty signatures in a given batch increases. This is the main advantage of this scheme as compared to Li et al. (2010) scheme. This scheme performs better in terms of number of modular exponentiations required for verification than Li et al. (2010) scheme for varied batch size, but if there are no faults in the batch then this method proves to be costlier than Harn (1998b) scheme.

(d) The technique by Seungwon et al. (2006) checks for the occurrence of bad signature using the Generic Test (GT) proposed by Bellare et al. (1998). And if there is presence of any bad signature/s, then the proposed technique can be applied to get the index of the bad signature. The technique belongs to Type 1 batch verification signing where all the messages are signed with a single private key. To find the index of the bad signature in a batch of signatures, first step is to compute $\prod_{i=1}^{t} m_i^i \bmod n$ and $\prod_{i=1}^{t} s_i^i \bmod n$. Then next step is to find k, an integer such that,

$$\left[\frac{(\prod_{i=1}^{t} s_i)^e}{(\prod_{i=1}^{t} m_i)}\right]^k = \frac{\left(\prod_{i=1}^{t} s_i^i\right)^e}{\prod_{i=1}^{t} m_i^i} \bmod n \tag{2.15}$$

If such an integer k exists foe Equation 2.15, then check the batch of signatures again using GT excluding k^{th} signature to check if there are any other bad signatures and if there are none then return k as the index of the bad signature. Finding the value of k is one of the time consuming task of this scheme. The method has increased number of modular exponentiations and modular multiplications.

2.1.2 DSS Batch verification schemes

National Institute of Standards and Technology (NIST) proposed DSS (Kravitz 1993) in 1991 and soon in 1993, it was regarded as Federal Information Processing Standard (FIPS) for digital signatures.

In this section, we discuss the different batch verification techniques for DSS. We also categorize these techniques based on whether the technique identifies only the

existence of a bad signature or also identifies the index of these bad signature.

(a) DSS Digital Signature Algorithm

DSS consists of three algorithms: key generation, signature generation, and signature verification. The three algorithms are briefed below:

1. Key Generation

- If l is the security parameter, then choose a large j-bit prime integer q and l-bit prime p such that $p - 1$ is a multiple of q.
- g is an order of q:
  - Select a random h, $0 < h < p$, and compute $g = h^{\frac{p-1}{q}} \bmod p$.
  - If $g = 1$, then re-select h.
- Choose a random integer x, $0 < x < q$ and calculate $y = g^x \bmod p$.
- Here x is the private key and y is the public key.

2. Signing

- For every message to be signed, select a random integer k, $0 < \mathrm{k} < q$.
- Compute $r = (g^k \bmod p) \bmod q$.
- Also compute $s = [k^{-1}(H(m) + xr)] \bmod q$.
- Signature is (r, s).

3. Verification

- Verify if $r' > 0$ and $s' < q$, if either of the conditions fail, reject the signature.
- Verifier receives (m', r', s'), and computes,

$$V = [g^{H(m')(s')^{-1} \bmod q} y^{r'(s')^{-1} \bmod q}] \bmod p$$

- $v = V \bmod q$
- If $v \stackrel{?}{=} r'$, then the signature is verified.

(b) Identifying existence of bad signature

In this section, we discuss the different batch verification techniques for DSS. We also categorize these techniques based on whether they identify only the existence of the bad signature or also identify the index of these bad signature/s. The techniques discussed in this subsection verify the existence of the bad signature/s in a given batch of digital signatures. If any of the signatures in the batch are faulty, then these techniques identify the batch as the invalid batch.

(a) Naccache et al. (1994) introduced two techniques: Interactive Batch Verification and Probabilistic Batch Verification. Both the techniques are efficient for Type 1 batch verification signing where single signer is involved in signing multiple messages.

In Interactive Batch verification, the signer chooses $k_i,\ k_i \in_R GF^*(q)$ and calculates and sends $r_i = g^{k_i} \bmod p$, for $1 \leq i \leq t$. The verifier replies with a message randomiser b_i of e-bit length for each message, which is appended to the message and then hashed before sending to the verifier. The signature generated is s_i and is given by $s_i = \frac{H(m_i|b_i)+xr_i}{k_i} \bmod q$

Then at the verifier, the verification Equation is 2.16

$$\prod_{i=1}^{t} r_i^{b_i} \bmod p \stackrel{?}{=} (g^{\sum_{i=1}^{t} w_i H(m_i|b_i) \bmod q})(y^{\sum_{i=1}^{t} w_i r_i \bmod q}) \bmod p \qquad (2.16)$$

where $w_i = \frac{1}{s_i} \bmod q$. This scheme requires computation of $t + 2$ modular exponentiations and $3t - 2$ number of modular multiplications for verification at the verifier. The size of the random variable b_i plays an important role in the security of the technique. Lim and Lee (1994) mainly discuss the security leak in Interactive Batch verification of DSS. They provide two instances: Directed chosen message attack by verifier and Generic chosen message attack by the signer.

(b) The second scheme of Naccache et al. (1994) is the probabilistic batch verification, where the signature generation for the message is same as the naive DSS algorithm, and the only difference is in the signature verification. At the verification, the verifier chooses pairwise relative prime equation

randomisers $b_1, b_2, \ldots, b_t \in_R GF^*(q)$ and verify the Equation 2.17,

$$\prod_{i=1}^{t} r_i^{b_i} \stackrel{?}{=} [(g^{\sum_{i=1}^{t} b_i w_i H(m_i) \bmod q})(y^{\sum_{i=1}^{t} b_i w_i r_i \bmod q})] \bmod p \qquad (2.17)$$

The technique requires $t+2$ modular exponentiations and $5(t-1)$ modular multiplications during verification. This technique provides security but at the cost of increased computational complexity.

(c) The scheme by Harn (1998a) for DSS-type algorithm is based on the different ElGamal type digital signature schemes (Harn and Xu 1994) and has the same parameters as the original DSS algorithm.

At the signature generation side,

$r = (g^k \bmod p) \bmod q$ and

$s = (rk - mx) \bmod q$

At the verification side,

$r = (g^{sr^{-1}} y^{mr^{-1}} \bmod p) \bmod q$

In order to verify t signatures $(r_1, s_1), (r_2, s_2) \ldots, (r_t, s_t)$ in batches, multiply all the r_i values of t signatures,

$$r_1 r_2 \ldots r_t \stackrel{?}{=} (g^{s_1 r_1^{-1} + s_2 r_2^{-1} \cdots + s_t r_t^{-1}} y^{m_1 r_1^{-1} + m_2 r_2^{-1} + \cdots + m_t r_t^{-1}} \bmod p) \bmod q \qquad (2.18)$$

From the above Equation 2.18, it is clear that to verify one signature in individual verification, two modular exponentiations are needed, and for t signatures, $2t$ modular exponentiations are needed. This batch verification technique requires just 2 modular exponentiations and $3t - 2$ modular multiplications to verify the batch of signatures. There are various instances (Hwang et al. 2001) where this technique fails to provide the expected security. This can be explained as follows,

Suppose (m_i, r_i, s_i) is generalised signature,

Let $s_i' = s_i + a_i r_i \bmod q$, where $\prod_{i=1}^{t} a_i = 0$, now the sender sends (m_i, r_i, s_i'). Therefore during batch verification, these multiple signatures satisfy the verification equation of the technique and hence it is difficult to identify forged invalid signatures. Such invalid signatures can be detected through individual verification but not by the above verification technique.

(d) The technique by Shao (2001) accounts for verification of different signatures by different signers for different messages and hence can be categorized under Type 3 way of signing for batch verification. $u_1 + u_2 + \ldots u_t$ are a set of signatures generated by t signers with public keys $y_1, y_2, \ldots, y_t$ respectively. Here u_1 signatures are $\{r_{11}, s_{11}\}, \{r_{12}, s_{12}\}, \ldots \{r_{1u_1}, s_{1u_1}\}$ that are generated for messages $m_{11}, m_{12}, \ldots, m_{1u_1}$ with private key x_1. Similarly the signatures $u_2, \ldots, u_t$ are generated with various private keys. The Equation 2.19 represents verification.

$$\prod_{i=1}^{t}\prod_{j=1}^{u_i} r_{ij}^{u_{ij}} \stackrel{?}{=} (g^{\sum_{i=1}^{t}\sum_{j=1}^{u_i} u_{ij}s_{ij}})(\prod_{i=1}^{t} y_i^{\sum_{j=1}^{u_i} u_{ij}H(m_{ij},r_{ij})}) \bmod p \qquad (2.19)$$

The main disadvantage of the scheme is increased number of modular exponentiation operations at the verification. Even if the security is comparable to individual verification, it has lengthy signatures which may lead to computation overhead. The scheme is expected to perform better for elliptic curve digital signature as per Shao (2001), since it can speed up verification by 50% in comparison to individual verification.

(e) In the technique proposed by Lin et al. (2005), batch verification time speeds up for DSS signatures. Batch verification procedure of this scheme belongs to Type 1 way of signing messages where the technique is efficient for single signer. The technique introduces a number $'S'$ at the verification side. The verification equation can be explained as follows,

Choose randomly $b_1, b_2, \ldots, b_t \in \{0,1\}^l$ and compute for $j = 1, 2, \ldots, t$

$$S_j = \prod_{i=1, i\neq j}^{t} s_i \bmod q$$

And also compute $S = (s_1 S_1) \bmod q$ and then verify,

$$(\prod_{i=1}^{t} r_i^{b_i} \bmod p)^S \bmod q \stackrel{?}{=} [(g^{\sum_{i=1}^{t} m_i S_i b_i \bmod q})(y^{\sum_{i=1}^{t} r_i S_i b_i \bmod q})] \bmod p \qquad (2.20)$$

If the batch verification Equation 2.20 holds true, then accept or else reject the signature. The verification scheme requires $l + t(7 + l/2) - 6$ number of modular multiplications and three modular exponentiations, where l is the

length of the random number b_i. This batch verification scheme removes the inverse modular exponentiation operation at the verifier but instead increases the number of modular multiplications.

Table 2.3: Comparative analysis of Batch verification techniques for DSS

Technique	Efficient for	Possibility of Forging	Forger	No. of Modular Exponentiations	Index of bad signature	Pros	Cons
Interactive (Naccache et al. 1994)	single signer	yes	signer, verifier	constant	no	Provides quick batch verification and reduces computation load.	Attacks are possible since the randomizers are generated at the signer.
Probabilistic (Naccache et al. 1994)	single signer	yes	signer	linear function of t	no	Introduction of randomizers at verifier increases the security.	The number of modular exponentiations increases due to the introduction of randomizers.
Harn (1998a)	single signer	yes	signer	constant	no	Number of exponentiations are reduced considerably.	It is proved to be erroneous and insecure.
Shao (2001)	multiple signers	no	-	linear function of t	no	Its security is comparable to individual verification.	The signature size and computation time are high.
Lin et al. (2005)	single signer	no	-	liner function of t (exponents are small)	no	Does not need modular inverse computation.	Increased number of modular multiplications.
Shen et al. (1999)	single signer	no	-	linear function of t	yes	Identifies the index of bad signature/s if the number of invalid signatures is known beforehand.	Efficiency decreases as the number of bad signatures increases.
Pastuszak et al. (2000a)	single signer	no	-	logarithmic function of t	yes	Very efficient if the number of bad signatures is known beforehand.	Efficiency reduces as the number of bad signatures increases.

Tables 2.3 and 2.4 discuss about the batch verification techniques for DSS. Table 2.3 makes a comparative study of the behavior of these different techniques. It also talks about the possibility of forgery by the signer and the pros and cons of the techniques. The various parameters of comparison are:

- Efficient for: Indicates whether the scheme is efficient for single or multiple

Table 2.4: Characteristics of Batch verification techniques in DSS

Technique	Authenticity	Integrity	Non-repudiation	Computation Overhead
Interactive (Naccache et al. 1994)	-	-	-	low
Probabilistic (Naccache et al. 1994)	✓	-	-	high
Harn (1998a)	-	-	-	low
Shao (2001)	✓	✓	✓	high
Lin et al. (2005)	✓	✓	✓	high
Shen et al. (1999)	✓	✓	✓	high
Pastuszak et al. (2000a)	✓	✓	✓	high

signers.

- Possibility of Forging: This parameter indicates whether the scheme is vulnerable to forgery attacks.
- Forger: This parameter indicates who can forge the signatures if there is a possibility of forgery.
- No. of Modular Exponentiations: This parameter provides the runtime analysis of the schemes. The variable t is the batch size.
- Index of bad signature: This parameter indicates whether the scheme identifies the location of the faulty signature.
- Pros & Cons: These columns indicate the strengths and weaknesses of the schemes.

Table 2.4 shows how the basic properties of the digital signatures are satisfied by different techniques of DSS. The computation overhead column indicates whether the scheme's batch verification time depends on the batch size. For certain schemes, the verification time increases as the batch size increases. Hence such schemes have high computation overhead.

(c) Identifying location of bad signature

In this category, we discuss the batch verification techniques proposed for DSS, which identify the existence of the bad signature along with the index of the bad signature. These techniques consume more time than the techniques proposed for identifying just the existence of the bad signature in a batch of signatures discussed in Section 2.1.2(**b**).

(a) The technique proposed by Shen et al. (1999) uses a binary tree-searching algorithm to find the invalid signature/s in the given batch. This technique is applied if the given batch fails the verification test for the techniques proposed by Harn (1998a). The given batch is divided into two sub-batches and again the same verification test is applied on the two sub-batches independently. This technique is efficient for Type 1 way of signing in batch verification. The algorithm takes the entire batch, its first and last indices, and status as inputs. The status stores the result status of the algorithm as either valid or invalid. After independently verifying the two sub-batches, the technique tries to identify which sub-batch is faulty by checking status and applies the same steps repeatedly to arrive at the bad signature. It uses divide and conquer approach. The proposed technique requires Min$(2t-1, 2k(\lceil \log t \rceil - \log k) + 2k - 1)$ verification tests, where k is the number of invalid signatures among the batch of t signatures. The performance of the technique ceases as the number of faults k increases within the batch. The algorithm is most efficient if k is less than $0.5t/\log t$.

(b) The Hamming Verifier (Pastuszak et al. 2000a) technique mainly focusses on identifying the location of bad signature in a batch of digital signatures. The technique establishes the relationship between the degree of contamination and the number of batch instances. The technique compares its efficiency with Divide and Conquer method (Pastuszak et al. 2000a; Seungwon et al. 2006) based on the number of GTs needed to identify the bad signature. Depending on the probability distribution, it is proved that the divide and conquer method is efficient for 2^k signatures, if there are 2^{k-3} or less number of bad signatures, and Hamming Verifier is efficient for a batch

of length $2^k - 1$ signatures if there is a single bad signature. And again if the number of bad signatures is known beforehand then divide and conquer becomes much more efficient, with almost the number of verification tests reduced by almost half. Hamming verifier technique is based on the parity check matrix, and it successfully identifies a single bad signature. And, if there are two bad signatures, then a two-layer Hamming Verifier is efficient in identifying the two signatures based on Bose, Chaudhuri, and Hocquenghem (BCH) (Forney 1965) codes accurately. Similarly, it can be extended to more than two bad signatures. But the efficiency of Hamming code verifier reduces as the number of bad signatures increases.

2.1.3 ECDSA Batch Verification schemes

There has been growing interest in ECDSA in recent years. ECDSA provides the same level of security as RSA with reduced key size. ECDSA is a digital signature algorithm of Elliptic Curve Cryptography (ECC) which is either based on DLP or Bilinear Pairing. The short signature techniques based on Bilinear Pairing use Elliptic Curve Groups. The following list briefs about the various techniques proposed for improving the batch verification of ECDSA.

(a) ECDSA Digital Signature Algorithm

ECDSA is a variant of Elliptic Curve Cryptosystem (Blake et al. 1999; Brown 2005; Koblitz 1987, 1998) similar to DSS. The parameters for computation are considered from the elliptic curve group $E(\mathbb{F}_q)$ (Brown 2005). It is currently IEEE P1363-2000, FIPS 186-3 standard. The digital signature generation and verification methods are defined in ANS X9.62. The curve parameters are chosen in such a way that the Elliptic Curve Discrete Logarithm Problem (ECDLP) can not be solved in less than exponential time.

The three algorithms: key generation, signature generation and signature verification are provided in this section. The ECDSA signature (r, s) has r and s values modulo n.

The standard equation for elliptic curve is given in Equation 2.21,

$$y^2 = x^3 + ax + b \tag{2.21}$$

where *a*, $b \in Z_p$ and $4a^3 + 27b^2 \neq 0 \bmod p$

Table 2.5: Notations followed in ECDSA

Symbol	Reference to
q	Order of the prime field $\mathbb{F}_q$.
$E(\mathbb{F}_q)$	Elliptic Curve defined over prime field $\mathbb{F}_q$.
n	Order of P, usually a prime.
P	Random non-zero base point of order n in $E(\mathbb{F}_q)$.
h	The cofactor $\frac{E(\mathbb{F}_q)}{n}$.
t	Size of the batch of signatures.

1. Key Generation

- Choose a non-zero point P, $P \in E(\mathbb{F}_q)$.
- Choose a random integer d such that $1 \leq d \leq n-1$.
- Compute $Q = dP \in E(\mathbb{F}_q)$.
- The Public key is (Q) and the private key is d.

2. Signing

- Signer chooses a random integer k, such that $1 \leq k \leq n-1$.
- Compute $R = kP$.
- Compute r as the x - coordinate of point R. $r = x(R) \bmod n$.
- Then compute $s = k^{-1}(H(m) + dr)$ where $H(m)$ is the hash value of the message m to be signed.
- Signature (r, s) is appended with the message and sent to verifier.

3. Verification

- The verifier receives (m', r', s') and verifies if $(r', s') \in [1, n-1]$, else reject the signature.
- Compute $w = (s')^{-1} \bmod n$.
- Then compute $u = H(m')w \bmod n$.
- Compute $v = r'w \bmod n$.

- $R = uP + vQ \in E(\mathbb{F}_q)$, and if $x(R) \stackrel{?}{=} r'$, then accept the signature or else reject.

(b) Signatures Based on DLP

In this subcategory, we analyze the techniques for batch verification of ECDSA signatures which are based on DLP. Most of the techniques discussed here are efficient for single as well as multiple signers.

(a) ECDSA* signature scheme proposed by Antipa et al. (2005) provides 40% efficiency in the verification time for ECDSA signatures. The security of their signature scheme is equivalent to ECDSA. The basic batch verification scheme for ECDSA* signatures is also known as the naive batch verification scheme and it belongs to Type 2 way of signing. The signature of modified ECDSA* signature scheme is (R, s), where R is the Elliptic curve point with (x, y) coordinates, whereas the original ECDSA signature is (r, s) with respect to (m, Q), where m is the message to be signed and Q is the public key. The scheme involves appending a few bits of side information to the signature by the signer. To improve the efficiency of original ECDSA signature (r, s), it is first converted to ECDSA* (R, s) signature. Hence reconstructing R from r in an efficient way to speedup ECDSA is important. Thus for an elliptic curve $E(F_q)$ over finite field F_q , with cofactor h, there can be $h + 1$ values for x coordinate of R. The number of $\lceil log_2(h + 1)\rceil + 1$ bits of side information can efficiently identify the point R. The extra overhead of side information is the disadvantage, but this extra information can help boost the verification time by 40%.

(b) The scheme proposed by Cheon and Yi (2007) claims to give better batch verification results for single as well as multiple signers. Hence it can be classified under Type 2 way of signing in batch verification. The scheme involves generation of width-w Non-Adjacent Forms (w-NAFs) uniformly. After the experimentation with different values of w, they arrived at optimum value of w as 3, which leads to a minimum number of

point doublings, point additions, and exponentiations during ECDSA batch verification at the verifier. For t signatures (m_i, R_i, s_i), where $R_i = (x_i, y_i)$, $r_i = x_i \bmod q$ and $s_i = k^{-1}(m_i + xr_i)$, the verification equation is $R = aG + bQ$ where $a = ms^{-1} \bmod q$ and $b = rs^{-1} \bmod q$. The modified ECDSA batch verification scheme is as follows: The technique uses random w-NAFs $c_1, c_2, \ldots, c_t$ and applies batch verification by 3-NAFs for multiple signers, where $a = -\sum_{i=1}^{t} a_i c_i \bmod q$ and $b'_i = -r_i s_i^{-1} c_i \bmod q$ for each i with digit set D$= \{\pm 1, \pm 3\}$, compute

$$aG + \sum_{i=1}^{t} b'_i Q_i + \sum_{i=1}^{t} c_i R_i \tag{2.22}$$

and for single signer, compute $a = -\sum_{i=1}^{t} a_i c_i \bmod q$ and $b = -\sum_{i=1}^{t} r_i s_i^{-1} c_i \bmod q$ and then compute,

$$aG + bQ + \sum_{i=1}^{t} c_i R_i \tag{2.23}$$

If the result of both the expressions 2.22 and 2.23 is equal to point at infinity O, then accept all the signatures of the batch, else reject the signatures. The overhead in this scheme is to convert the original ECDSA signature to modified ECDSA* signature.

(c) Two batch verification techniques for ECDSA and the naive verification are introduced by Karati et al. (2012b): one is the naive approach and the other two are efficient algorithms **Algorithm S1, Algorithm S2** which perform better than naive approach. All the three techniques belong to Type 2 category of signing for batch verification.

In case of naive approach, for the verification of signatures by multiple signers is given in the Equation 2.24,

$$\sum_{i=1}^{t} R_i \stackrel{?}{=} \left(\sum_{i=1}^{t} u_i\right) P + \sum_{i=1}^{t} v_i Q_i \tag{2.24}$$

In case of single signer, the verification equation is Equation 2.25,

$$\sum_{i=1}^{t} R_i \stackrel{?}{=} \left(\sum_{i=1}^{t} u_i\right) P + \left(\sum_{i=1}^{t} v_i\right) Q \tag{2.25}$$

This involves computation of modular square roots for the value of y, which is a time-consuming operation. The modified algorithm for the same would

be the ECDSA*, which includes addition of a few bits in signature which avoids the calculation using square root operation.

Algorithm S1 does not use the regular method of finding y using square root technique. As we know, $R_i = (x_i, y_i)$ and $r_i = x(R_i)$ ie., $x_i = r_i$. The value of y_i is calculated using the Equation 2.26,

$$y_i^2 = r_i^3 + ar_i + b \bmod q \tag{2.26}$$

Now by applying multiple elliptic-curve point addition formula for point $R = \sum_{i=1}^{t} R_i$, which results in,

$$R = \left(\frac{g_x(y_1, y_2, \ldots, y_t)}{h_x(y_1, y_2, \ldots, y_t)}, \frac{g_y(y_1, y_2, \ldots, y_t)}{h_y(y_1, y_2, \ldots, y_t)} \right)$$

where the denominator is $u(y_2, y_3, \ldots, y_t)y_1 + v(y_2, y_3, \ldots, y_t)$. In order to eliminate y_1, multiply both g_x and h_x by $u(y_2, y_3, \ldots, y_t)y_1 - v(y_2, y_3, \ldots, y_t)$, which eliminates y_1 using the formula stated in Equation 2.26, and using Equations 2.24 and 2.25, we express $R = (\alpha, \beta)$ where $\alpha, \beta \in \mathbb{F}_q$,

$$R_x(y_1, y_2, \ldots, y_t) = \alpha \tag{2.27}$$

$$R_y(y_1, y_2, \ldots, y_t) = \beta \tag{2.28}$$

Then the multivariate Equations 2.27 and 2.28 are solved to retrieve the value of y. They avoid the calculation of u and v from the basic equation. Since the algorithm gives effective results for batch size $t \leq 6$, its efficiency decreases as we increase the batch size. And also in case of point addition, if the two points turn out to be equal, then the results will be different since there is a different formula for point doubling in case of elliptic curve cryptosystems.

(d) **Algorithm S2** is similar to Algorithm S1, but the procedure for solving the multivariate equations to get the values for x and y are modified to reduce the complexity of finding the sign of y.

In this algorithm, only the first multivariate equation is considered where the $y_1, y_2, \ldots, y_t$ are eliminated through an equation $u_i^2(r_i^3 + ar_i + b) - v_i^2$ where y_i is eliminated and u, v are polynomials in $y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_t$. And if the equation reduces to zero, then the original equation is consistent

with the equation for ECDSA.

Table 2.6: Comparative analysis of Batch verification techniques for ECDSA

Technique	Efficient for	Conversion to ECDSA*	Efficient for Batch size	Pros	Cons
Naive ECDSA	single signer	no	$t < 6$	Most secured technique.	Includes heavy calculation for finding the square roots of the equation.
Naive ECDSA* (Antipa et al. 2005)	multiple signers	yes	$t > 8$	It is one of the most efficient ones.	The inclusion of side information bits increases size of signature.
Cheon and Yi (2007)	multiple signers	yes	any t	Algorithm is efficient for single and multiple signers.	It introduces precomputation of w-NAF.
Algorithm S1 (Karati et al. 2012b)	single signer	no	$t < 6$	It is quite efficient for small batch size.	Its efficiency suddenly decreases with the increase in batch size.
Algorithm S2 (Karati et al. 2012b)	single signer	no	$t \leq 8$	Its efficiency is comparable to ECDSA*.	Performance deteriorates as the curve field increases.

Table 2.7: Characteristics of Batch verification techniques in ECDSA

Technique	Authenticity	Integrity	Non-repudiation	Speed-up
Naive ECDSA	-	✓	✓	low
Naive ECDSA* (Antipa et al. 2005)	-	✓	✓	high
Cheon and Yi (2007)	✓	✓	✓	high
Algorithm S1 (Karati et al. 2012b)	✓	✓	✓	low
Algorithm S2 (Karati et al. 2012b)	✓	✓	✓	low

The two Tables 2.6 and 2.7 analyze the ECDSA batch verification techniques. Table 2.6 discusses the behavior of these different techniques. This table also

shows the pros and cons of the techniques. The various parameters can be explained as:

- Efficient for: Indicates whether the scheme is efficient for single or multiple signers.
- Conversion to ECDSA*: This indicates whether the scheme first converts the signatures to ECDSA* to make the computation easier.
- Efficient for Batch Size: This indicates the batch size for which the scheme performs its best.
- Pros & Cons: These columns indicate the strengths and weaknesses of the schemes.

And in Table 2.7, we summarize how these techniques satisfy the digital signature properties. The speedup parameter compares the speedup gained by the technique compared to the individual verification. The lower range is the speedup in case of multiple signers, and the higher range is for the single signer.

(c) Signatures based on Bilinear Pairing

In this section, we review the batch verification schemes for short signatures (Boneh and Boyen 2004; Boneh et al. 2004; Delerablée and Pointcheval 2006) based on Bilinear Pairing (Koblitz and Menezes 2005). There are two variations to this kind of signatures. One is batch verification scheme for signatures either belonging to Type 1, Type 2 or Type 3. The other one is the batch verification scheme for group signatures.

Regular Signature: In regular signature category, we discuss the batch verification techniques based on Bilinear pairing, which are used to verify signatures generated either by Type 1, Type 2, or Type 3.

(a) The batch verification technique by Camenisch et al. (2007) is introduced for the Boneh-Lynn-Shacham (BLS) signature scheme defined by Boneh et al. (2001). First, we will describe the signature scheme, which is based

on bilinear pairing.

1. **Key Generation:**
   - Choose a random integer $x \in \mathbb{Z}_q$, where q is the prime order of group $\mathbb{G}$.
   - Private Key $sk = x$ and public key $pk = g^x$, where g is the generator of group $\mathbb{G}$.
   - And $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ be the pairing map which satisfies the properties of Bilinearity and Non-degeneracy.
2. **Signing**
   - Calculate hash value for the message m, $M = H(m)$.
   - Given the private key sk, calculate the signature $s = M^x$.
3. **Verifying**
   - Verifier verifies the signature by
   $$e(s, g) \stackrel{?}{=} e(M, g^x) \tag{2.29}$$
   - If LHS is equal to RHS, then the signature is successfully verified.

Batch Verification for BLS Signature Scheme: The batch verification technique proposed for BLS by Camenisch et al. (2007) belongs to Type 2 way of batch verification where the verifier verifies signatures signed by multiple signers. The verification equation for the same is given by:

$$e(\prod_{i=1}^{t} s_i^{\delta_i}, g) \stackrel{?}{=} \prod_{i=1}^{t} e(M_i, pk_i)^{\delta_i} \tag{2.30}$$

where s_i are the signatures of t signers for t distinct messages with t public keys pk_i, where $1 \leq i \leq t$. δ_i is a random element of l bits from $\mathbb{Z}_q$ and δ is the vector defined as $\delta = \delta_1, \delta_2, \ldots, \delta_t$. Once the verifier receives the signatures, it first verifies if all the public keys pk_i are valid, and all the signatures $s_i \in \mathbb{G}$ for all i, otherwise reject. If the above Equation 2.30 holds then accept all the signatures in the batch, otherwise it indicates the presence of invalid signature/s. The computation time increases linearly with the number of signers. Hence the scheme is inefficient for large number

of signers.

(b) The Camenisch and Lysyanskaya (CL) signature scheme proposed by Camenisch and Lysyanskaya (2004) is not efficient for batch verification. Therefore, the modified version of the CL signature scheme CL* is proposed by Camenisch et al. (2007), which is suitable for batch verification is presented below.

1. Key Generation

- Choose x randomly, such that $x \in \mathbb{Z}_q$ and then calculate $X = g^x$, where g is the generator for the group $\mathbb{G}$ of prime order q.
- $sk = x$ and $pk = X$.

2. Signing

- $\varphi \in \phi$ is the current time period of generating signatures where $|\phi|$ is the polynomial.
- Compute $w = H_3(m||\varphi)$, $a = H_1(\varphi)$, $b = H_2(\varphi)$ where H_1, H_2 and H_3 are the hash functions $H_1 : \phi \rightarrow \mathbb{G}$, $H_2 : \phi \rightarrow \mathbb{G}$ and $H_3 : M \times \phi \rightarrow \mathbb{Z}_q$, where $M = \{0,1\}^*$ is the message space and the message $m \in M$.
- Signature $s = a^x b^{xw}$.

3. Verifying

- After receiving message-period pair (m, φ) and signature s, compute $w = H_3(m||\varphi)$, $a = H_1(\varphi)$ and $b = H_2(\varphi)$.
- Then verify if the equation given below holds,

$$e(s', g) \stackrel{?}{=} e(a, X).e(b, X)^w \tag{2.31}$$

- If this is true, then the signature is accepted or else rejected.

Batch Verification for CL* Signature Scheme: Camenisch et al. (2007) proposed a technique for batch verification of CL* Type 2 signatures. The t signatures $s_1, s_2, \ldots, s_t$ are generated for the messages $m_1, m_2, \ldots, m_t$ using the public keys $X_1, X_2, \ldots, X_t$ at the same time period φ. Once the signatures are received at the verifier, first is to verify if X_i are valid and

$s_i \in \mathbb{G}$, where $i \in [1, \ldots, t]$, otherwise reject the batch of signatures. Next is to verify the equation,

$$e(\prod_{i=1}^{t} s_i^{\delta_i}, g) \stackrel{?}{=} e(a, \prod_{i=1}^{t} X_i^{\delta_i}).e(b, \prod_{i=1}^{t} X_i^{w_i \delta_i}) \tag{2.32}$$

where δ_i is the randomly generated number of l bit length. If this equation holds then accept all the signatures in the given batch or else reject.

(c) The Guo-Mu-Chen (GMC) signature scheme which is introduced by Guo et al. (2008) is another batch verification scheme based on bilinear pairing. The GMC scheme for individual signature is defined as follows:

1. Key Generation

- $\mathbb{G}_1$ and $\mathbb{G}_2$ be the bilinear groups with prime order q and randomly generate $g_2 \in \mathbb{G}_2$ and calculate $g_1 = \psi(g_2)$.
- Generate $\alpha, \beta \in \mathbb{Z}_q$.
- Compute $u = e(g_1, g_2) \in \mathbb{G}_T$, $v = e(g_1, g_2)^{\alpha} \in \mathbb{G}_T$ and $z = g_2^{\beta} \in \mathbb{G}_2$.
- Public key $pk = (e, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, u, v, z)$ and secret key $sk = (\alpha, \beta)$.

2. Signing

- Generate a random integer $r \in \mathbb{Z}_q$ for message $m \in \mathbb{Z}_q$.
- Generate the signature $s_m = (s_1, s_2) = (\alpha - r(\beta - m), g_1^r) \in \mathbb{Z}_q \times \mathbb{G}_1$.

3. Verifying

- Verifier verifies the message m using the public key pk using the verification equation:

$$u^{s_i'}.e(s_2', z/g_2^m) \stackrel{?}{=} v \tag{2.33}$$

- If verification holds true, then the signature is valid.

Batch Verification for GMC signatures: This batch verification technique Guo et al. (2008) comes under Type 1 of batch signatures. For t signatures $s_{m_1}, s_{m_2}, \ldots, s_{m_t}$ for messages $m_1, m_2, \ldots, m_t$ signed using

sk, and the random integers generated are $r_1, r_2, \ldots, r_t \in \mathbb{Z}_q$. First step for batch verification is to check whether $g_1^{r_1}, g_1^{r_2}, \ldots, g_1^{r_t} \in \mathbb{G}_1^n$. Next is to choose a random vector $\delta = \delta_1, \delta_2, \ldots, \delta_t$, where the length of each δ_i is l bits. Then is to compute A, B, and C.

$A = (g_1^{r_1})^{\delta_1}.(g_1^{r_2})^{\delta_2} \ldots (g_1^{r_n})^{\delta_t}$

$B = (g_1^{r_1})^{m_1\delta_1}.(g_1^{r_2})^{m_2\delta_2} \ldots (g_1^{r_n})^{m_n\delta_t}$

$C = \delta_1(\alpha - r_1(\beta - m_1)) + \delta_2(\alpha - r_2(\beta - m_2)) + \cdots + \delta_t(\alpha - r_t(\beta - m_t))$

$= \alpha \sum_{i=1}^{t} \delta_i - \beta \sum_{i=1}^{t} \delta_i r_i + \beta \sum_{i=1}^{t} \delta_i r_i m_i.$

The verification equation is given by Equation 2.34,

$$\frac{u^C.e(A,z)}{e(B,g_2)} \stackrel{?}{=} v^{\sum_{i=1}^{t}\delta_i} \tag{2.34}$$

Group Signature Generation: This section explains the batch verification scheme of group signatures (Boneh et al. 2004) based on Bilinear pairing. In the case of group signatures, a single member of the group generates signature for the message on behalf of the entire group. The identity of the signing member is kept hidden and is only known to the group manager who can identify every member of the group through unique identification key.

The algorithm to verify a group signature can be given as follows:

Let $PSetup(1^\tau) \rightarrow (q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, where $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$ is a hash function, g_1, g_2 are generators of groups $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively and there exists an efficiently-computable isomorphism $\psi : \mathbb{G}_2 \rightarrow \mathbb{G}_1$. Let l be the number of users in a group.

1. **Key Generation**

   - Randomly select $g_2 \in \mathbb{G}_2$ and also sets $g_1 \leftarrow \psi(g_2)$.
   - Choose $h \stackrel{R}{\leftarrow} \mathbb{G}_1 \backslash \{1_{\mathbb{G}_1}\}$, $r_1, r_2 \stackrel{R}{\leftarrow} \mathbb{Z}_q^*$ and also set values of u, v such that, $u^{r_1} = v^{r_2} = h$.
   - Choose $\gamma \stackrel{R}{\leftarrow} \mathbb{Z}_q^*$ and assign $w = g_2^\gamma$.
   - For $i = 1$ to n, choose $x_i \stackrel{R}{\leftarrow} \mathbb{Z}_q^*$ and also set $f_i = g^{1/(\gamma + x_i)}$.
   - Public key is (g_1, g_2, h, u, v, w), the group manager's secret key is (r_1, r_2) and the ith user secret key is (f_i, x_i).

2. Signing

- Choose $\alpha, \beta, r_\alpha, r_\beta, r_x, r_{\gamma_1}, r_{\gamma_2} \xleftarrow{R} \mathbb{Z}_q$.
- Compute $T_1 = u^\alpha$; $T_2 = v^\beta$; $T_3 = f.h^{\alpha+\beta}$; $\gamma_1 = x.\alpha$; $\gamma_2 = x.\beta$; $R_1 = u^{r_\alpha}$; $R_2 = v^{r_\beta}$; $R_3 = e(T_3, g_2)^{r_x}.e(h, w)^{-r_\alpha - r_\beta}.e(h, g_2)^{-r_{\gamma_1} - r_{\gamma_2}}$; $R_4 = T_1^{r_x}.u^{-r_{\gamma_1}}$; $R_5 = T_2^{r_x}.v^{-r_{\gamma_2}}$.
- Compute $c = H(m, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$.
- Compute $s_\alpha = r_\alpha + c.\alpha$; $s_\beta = r_\beta + c.\beta$; $s_x = r_x + c.x$; $s_{\gamma_1} = r_{\gamma_1} + c.\gamma_1$; $s_{\gamma_2} = r_{\gamma_2} + c.\gamma_2$.
- Generate the signature $\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\gamma_1}, s_{\gamma_2})$.

3. Verifying

- Compute $R_1 = u^{s_\alpha}.T_1^{-c}$; $R_2 = v^{s_\beta}.T_2^{-c}$; $R_3 = e(T_3, g_2)^{s_x}.e(h, w)^{-s_\alpha - s_\beta}.e(h, g_2)^{-s_{\gamma_2} - s_{\gamma_2}}$; $R_4 = T_1^{s_x}.u^{-s_{\gamma_1}}$; $R_5 = T_2^{s_x}.v^{-s_{\gamma_2}}$.
- Verify if $c' \stackrel{?}{=} H(m, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$.

Batch Verification of BBS Group Signatures: Boneh - Boyen - Shacham (BBS) Scheme is the group signature scheme introduced by Boneh et al. (2004). Here the message is signed by one of the members of the group whose identity is hidden. Only the group manager can trace back the actual signer in the group through the group secret key gsk. The BBS scheme key generation, signature generation, and verification details are explained in Boneh et al. (2004).

For batch verification of BBS Group Signatures (Boneh et al. 2001), a small modification is done to original BBS signature, making it suitable for batch verification. The most expensive operation in signature verification is the computation of R_3. The batch verification of BBS group signatures as stated by Ferrara et al. (2009), for m signatures, involves the addition of an extra element in the signature, which increases signature size.

For $j = 1, 2, \ldots, t$ signatures, compute the R_1, R_2, R_4, R_5 for every signature. Then for every $j = 1, 2, \ldots, t$ signature, verify if c_j = $H(M_j, T_{j,1}, T_{j,2}, T_{j,3}, R_{j,1}, R_{j,2}, R_{j,4}, R_{j,5})$.

Then at the end, with just two Bilinear pairings, equation verifies the batch of signatures,

$$e(\prod_{j=1}^{t}(T_{j,3}^{s_{j,x}}.h^{-s_{j,\gamma 1}-s_{j,\gamma 2}}.g_1^{-c_j})^{\delta_j}, g_2).e(\prod_{j=1}^{t}(h^{-s_{j,\alpha}-s_{j,\beta}}T_3^c)^{\delta_j}, w) \stackrel{?}{=} \prod_{j=1}^{t} R_{j,3}^{\delta_j} \tag{2.35}$$

where t random vectors $\delta_1, \delta_2, \ldots, \delta_t$ of l bit length $\in Z_q$. Accept the batch of signatures only if the equation holds correct.

Table 2.8: Comparative study of Batch verification schemes based on Bilinear Pairing

Technique	Efficient for	Type of Signature Generation	No. of Bilinear Pairings	Security using Random Oracle Model	Pros	Cons
BLS (Camenisch et al. 2007)	single signer	Individual Signature	t+1(multi-signer) 2(single-signer)	yes	For single signer, the number of pairings is just 2 and security level is comparable to RSA - 1024.	For multiple signers, the verification time increases.
II - Sig (Camenisch et al. 2007)	multiple signers	Individual Signature	3	yes	The scheme is more secure and is suitable for multiple signers.	The scheme has time period restrictions for the signer.
GMC (Guo et al. 2008)	single signer	Individual Signature	t+1(muti-signers) 2(single signer)	no	Efficient as BLS scheme (Camenisch et al. 2007) without random oracle.	The scheme does not provide any improvement in verification efficiency over existing schemes and has a fixed signature size of 320 bits.
BBS (Ferrara et al. 2009)	multiple signers	Group Signature	2	yes	The no. of pairings is constant and is independent of no. of signers.	The scheme increases the signature size of each signature to make it compatible for batch verification.

Tables 2.8 and 2.9 make a comparative study on the behavior of Bilinear pairing based batch verification schemes using Elliptic Curve Groups. The parameters used for comparison of the schemes are:

- Efficient for: Indicates whether the scheme is efficient for single or multiple signers.
- Type of Signature Generation: Indicates whether the signature represents an

Table 2.9: Characteristics of Batch Verification techniques based on Bilinear Pairing

Technique	Authenticity	Integrity	Non-repudiation	Computation Overhead
BLS (Camenisch et al. 2007)	✓	✓	✓	high
II - Sig (Camenisch et al. 2007)	✓	✓	✓	low
GMC (Guo et al. 2008)	✓	✓	✓	high
BBS (Ferrara et al. 2009)	✓	✓	✓	low

individual entity or the group.

- No. of bilinear pairings: Provides the information on whether the number of bilinear pairing operations depends on the batch size or fixed.
- Security using Random Oracle Model: Specifies whether the scheme is secure when used with ransom oracle model.
- Pros & Cons: These columns indicate the strengths and weaknesses of the schemes.

The computation overhead is determined by the number of Bilinear pairings in the verification of signatures. For a batch of signatures with size t , the entries in the Table 2.9 for computation overhead can be interpreted as: High (When the number of Bilinear Pairings is a linear function of t), and for Low (When the number of Bilinear Pairings is a constant and independent of t).

(d) Identifying location of bad signatures in ECDSA

The batch verification schemes in ECDSA does not identify the invalid signature's location. Hence to identify the bad signature, one has to either perform individual verification, Divide-and-Conquer(DC) verification or verification using Hamming code verifier (Pastuszak et al. 2000a). These bad signature identification schemes are not specific to the signature scheme. It can be used with RSA, DSS, or ECDSA. There are other schemes by Ren et al. (2015) and Li et al.

(2010), which can be extended to various batch verification schemes. Different schemes have various disadvantages in real-time implementation. DC verifier cannot perform the best in all cases; its efficiency decreases as the number of faulty signatures increases. Hamming code verifier needs the information of the number of faulty signatures in the batch before verification, which is impossible to know before verification.

The schemes proposed by Li et al. (2010) and Ren et al. (2015) are compute intensive. Hence there is a need for faster verification scheme which can identify the location of the bad signature efficiently in less time compared to the existing scheme. Hence to implement batch verification in IoT, we need secure and efficient batch verification schemes which are faster and also can identify the location of the faulty signatures.

Most of the verification schemes discussed in Section 2.1 do not identify the bad signature in the batch of signatures. Therefore there are many bad signature identification schemes designed to locate the bad signature in a batch that fails the batch verification test. Hence in the next section, we discuss various bad signature identification schemes.

2.2 BAD SIGNATURE IDENTIFICATION SCHEMES

There have been many batch signature verification techniques introduced. Few techniques trade the security to achieve greater computation efficiency in verifying the signatures (Kittur and Pais 2017). But most of the techniques fail to locate the index of the bad signature/s in case if the batch verification fails.

There are multiple batch verification techniques introduced for various digital signature schemes (Fiat 1989; Harn 1998b; Karati et al. 2012b; Min-Shiang et al. 2001) which locate the index of the invalid signature in case of the failure of batch verification. Pastuszak et al. (2000a) makes a comparative study of Divide-and-Conquer (DC) approach and the naive approach. Some of the important definitions are given below:

Table 2.10: Various Bad Signature Identification Schemes

Technique	Prior Info.	No. of GT	Restriction on # bad signatures	Characteristics	Drawback
Sequential	No	Fixed	No	Performs verification test on every signature to identify the faulty one.	It is the most time taking verification process.
DC Verifier (Pastuszak et al. 2000a)	No	Variable	No	Proves to be efficient in average case where few of the signatures are bad in the batch.	Performs worst when all the signatures are bad in the batch.
DBI_{basic} (Seungwon et al. 2006)	Yes	Fixed	Yes	Very efficient in identifying the index of the bad signature.	Is efficient only if the batch contains only one bad signature.
IC Verifier (Pastuszak et al. 2000b)	Yes	Fixed	No	For better identification capability, uses indices of Hamming weight.	Every increase in the number of bad signatures, increases the computation.
Li et al. (2010)	No	Variable	Yes	Divides the given batch instance into MxN matrix and identifies the location of bad signature accurately.	Increases the number of computations considerably.
Ren et al. (2015)	No	Variable	No	Divides the given batch into MxMxM cube. This further reduces the memory size.	With the increase in the dimension, the number of GTs will also increase.

Table 2.10 makes a comparative analysis of the existing schemes to identify the index of the bad signature in a given batch of signatures. We have compared these existing techniques through various parameters. These parameters can be explained as below:

- *Prior Info:* This parameter indicates whether the technique needs prior knowledge of the number of bad signatures present in the given batch of signatures. Hence this helps in deciding the kind of environment this scheme can be applied.
- *No. of GT:* This parameter specifies whether the number of GT operations depends on the size of the batch. Therefore in a few techniques, if the number of GT is fixed, it means that the number of GT operations does not vary with the increase in the batch size.
- *Restriction on # Bad Signatures:* This parameter indicates whether there is a

restriction on the number of bad signatures in a given batch. Certain schemes do not give proper results if the number of bad signatures is more than one.

The techniques Li et al. (2010); Ren et al. (2015) use two order and three order matrix respectively to identify the bad signature in the batch. The given batch size of t signatures is placed in a matrix of order ($M \times N \geq t$). Then the GT verifier is applied on each row and column to identify the bad signature.

In the other technique Ren et al. (2015), the given batch of signatures is divided into a cube $M \times M \times M \geq t$. The signatures are distributed uniformly in the matrix, and computation is carried out similarly as the previous one.

To implement batch verification in IoT, we first have to study various existing efficient popular trust models.

2.3 SECURITY TRUST MODELS

In this section, we are surveying various security trust models, models used for efficient energy harvesting in sensor nodes. There are various trust models available which prefer the selection of entities based on the trust value of the entity in the cloud platform (Manuel 2015; Xiong and Liu 2004). There are other reputation based models which consider the reputation of the entity with their neighbours and also examines the entity's performance in the past (Kalra and Sood 2015; Selcuk et al. 2004; Vu et al. 2012) before considering as trusted. Most of the reputation based models are designed for Peer-to-Peer (P2P) network (Kamvar et al. 2003; Xiong and Liu 2004; Zhou and Hwang 2007). The nodes are trusted based on the feedback received from their peers. The trust can also be computed using the entity's behavior, its belief, and other parameters, as shown in Zhiwei et al. (2015). Buzzanca et al. (2017) considers even aging as a metric for trust computation in their study. Aging refers to the pattern of trust growth for the entity, which also takes into account the recent change in behavior of the entity.

Implementing authentication schemes in IoT have challenges such as low computing power and memory of sensor nodes. The sensor nodes also have the restriction of low battery capacity. Hence there are various research works on the energy

harvesting in sensor nodes and efficient usage of energy (Escolar et al. 2014; Fisher et al. 2015; Wu et al. 2017) as well as developing lightweight schemes (Wu et al. 2018). Some researchers are working on how efficient public key cryptographic schemes are in IoT and WSN environment (Kayalvizhi et al. 2010; Wander et al. 2005). Since individual signature verification is a costly affair in IoT, the batch signature verification reduces the computation load significantly. In the model by Kittur et al. (2017), the gateway node reduces the burden of computation by distributing its load to other nodes. The authors show that the load is distributed among other gateway nodes to reduce the burden of computation at a single node. But the disadvantage with this scheme is that the scheme assumes that the Gateway nodes are idle and are available most of the times.

There are many trust models developed for various type of networks. In the case of e-Bay transactions, user feedback is very important. Therefore in Peer-to-Peer (P2P) network, it is important to trust the feedback received from various users. There are various trust models available for Wireless Sensor Networks (WSN) for finding the trusted path for the information to reach the central hub. Hence we have provided a comparison of a few popular existing trust models.

Table 2.11: Trust Models

Scheme	Type of Network	Reputation	Feedback From	Save Trust Value of	Physical State of the Node
Eigen (Kamvar et al. 2003)	P2P	Distributed	Peers	All peers	No
Peer Trust (Xiong and Liu 2004)	P2P	Distributed	Peers	Neighboring Peers	No
BTRM-WSN (Mármol and Pérez 2011)	WSN	Distributed	Peers	Neighboring Peers	No
Power Trust (Zhou and Hwang 2007)	P2P	Centralized	Reputation System	Only one	No
ATSN (Chen et al. 2007)	WSN	Centralized	Agent	Only one	Yes
Ganeriwal et al. (2008)	WSN	Distributed	Peers	All Peers	No

We have compared various trust models in Table 2.11 with the proposed trust model using certain parameters as explained below:

- *Type of Network*: Type of network for which the model is developed, is an important parameter, which decides the way the trust model has to be developed efficiently. In the case of a P2P network, each node has sufficient computation capability and memory storage available to compute and store the trust value of

other peer nodes.

- *Reputation*: This parameter indicates whether the reputation of the node is decided at the single point of control or is decided collectively by the peers in the distributed network. We can observe in Table 2.11 that few of the models are centralized and few are distributed.

- *Feedback From*: The third parameter indicates the source from where any node gets its trust value. But the trust value in a centralized network can also be calculated from using the feedback received from the other peer nodes. But only a single entity decides the final trust value for the node.

- *Save Trust Value*: This parameter depends on the memory capacity of the node. In a few models, nodes store the trust value of all the peers in the network or store only the trust value of the neighboring peers. In centralized reputation models, the nodes need not have to store the trust value of other peers.

- *Physical State of the Node*: This parameter indicates whether the model considers the physical state of the node into account while choosing it. The physical state indicates whether the node is up and running, or whether the node is busy. This parameter is important in WSN and IoT environment.

The design of the trust model varies based on the application and network where the model needs to be deployed. Every application has its security and privacy requirements; hence accordingly, the models are designed. We can observe from Table 2.11 that our model is better suited for the IoT environments since it is centralized. Sensor nodes have low computation capability, and hence, a distributed model will not be suitable in such a case. Since we are considering a centralized model, the peers do not participate in the computation of trust value. Hence any sensor node gets its QoS value computed by the single entity, which saves the trust value of all the peers in the network. There is one more model, ATSN by Chen et al. (2007), which is also suitable for our network, with a difference that an extra entity Agent is required, that collects the feedback from the peer nodes in ATSN. Every node gets its QoS value finalized based on the final evaluation by the Agent.

2.4 RESEARCH GAPS

In the previous sections, we have surveyed various batch verification techniques for RSA, DSS, and ECDSA and have given a brief overview of the techniques and the various attacks possible on the techniques under different conditions. Therefore this poses the following Research challenges:

1. *Batch Verification for Multiple Signers:* Most of the techniques, as per our knowledge are efficient for Type 1 way of signing where single signer either signs a single message or multiple messages. Therefore there is a need for various batch verification techniques for RSA, DSS and ECDSA digital signatures which consider efficiency for multiple signers into account while designing new techniques.

2. *Reduced number of modular exponentiations/ Scalar Multiplications:* DSS batch verification techniques which identify the index of the bad signature requires minimum t number of modular exponentiations, where t is the size of the batch of DSS digital signatures. Therefore developing techniques which identify the index of the bad signature in less than t number of modular exponentiations is a major challenge.

3. *Ensuring Non-repudiation:* Providing non-repudiation property for the digital signatures is one of the major challenges in batch verification of digital signatures. As we have observed, most of the techniques fail in satisfying this criterion of digital signatures where the signer denies of signing the message.

4. *Efficiency, irrespective of batch size:* Most of the batch verification techniques introduced for ECDSA are efficient for small batch size. Therefore in practical implementation, a technique becomes more advantageous if it performs well for varied batch size.

5. *Minimizing precomputation in ECDSA:* Since ECDSA is grabbing attention in recent times due to its short signature property, it has the overhead of precomputation. ECDSA* is one of the ways to minimize the precomputation

overhead, but it increases the signature size to gain efficiency. Although the side bit information in ECDSA* does not create significant overhead but developing techniques as efficient as or more efficient than ECDSA* requires attention.

6. *Provable Security using Random Oracle Model:* In case of batch verification schemes based on Bilinear pairing, it is important to develop schemes which are efficient for varied batch size with the provably secure Random Oracle Model. It is difficult for any application to practically implement the Random Oracle model that covers all security properties of the model.

7. *Bad Signature Identification in a batch:* The existing schemes have disadvantages such as poor performance if the number of faulty signatures is more. One more downside is the pre-knowledge of the number of bad signatures before verification. Other techniques have compute-intensive operations which hinder the performance of the scheme. Hence there is a need for lightweight and secure scheme which performs consistently irrespective of the number of bad signatures.

8. *Trust Model for IoT:* IoT nodes are more vulnerable to various kinds of attacks because of their low computation power and energy. Hence designing a trust model which can reduce the computation complexity without significantly affecting the life of the node is important.

2.5 SUMMARY

In this chapter, we have provided an in-depth survey of various batch verification schemes and various trust models. We have categorized the batch verification schemes based on the digital signature for which they are proposed. We have also made a comparative study of these schemes. This survey has led us to the discussion of various unresolved problems and issues which need attention from the majority of researchers. We also studied various bad signature identification schemes available in the literature. We also have studied a few popular trust models for P2P network, WSN network is also discussed and compared to identify the suitability for IoT network.

Chapter 3

BATCH VERIFICATION OF ECDSA* SIGNATURES

As we are designing a batch signature verification scheme for IoT network, it is essential to have a scheme which is lightweight as well as efficient for different batch size. Therefore we have decided ECDSA* as the digital signature algorithm for our study since it accelerates verification by 40% as well as it has small key size. Since ECDSA* signature verification is faster compared to other schemes studied in Chapter 2, we have designed a new batch verification scheme for these signatures. The only disadvantage of ECDSA* signatures is their increased signature size. But these extra bits do not make any changes to the standard ECDSA specifications. Hence ECDSA* signatures are in conformant with the ECDSA standards.

We have designed a batch verification scheme for ECDSA* signatures which can also be applied on ECDSA signatures too. The primary advantage of our scheme is that the efficiency of the scheme does not vary significantly with the increase in the batch size. The other highlighting factor of our scheme is the inclusion of random pairwise primes which enhance the security of the scheme further.

We have analysed various disadvantages and attacks possible on the existing schemes in the literature. Our scheme is inspired by the work of Naccache et al. (1994), which is proposed for DSS signatures. The scheme for DSS is secure against many attacks by the unauthorized user and also the fraud signer.

The contributions of the chapter towards research include:

- A new batch verification scheme for verifying ECDSA* signatures is proposed, which is efficient for various batch sizes.
- Detailed security analysis of the proposed scheme is provided.
- Cost and complexity analysis of the proposed scheme is provided.

The rest of this chapter is organized as follows: Section 3.1 provides definitions and notations; in Section 3.2, we introduce our new batch verification algorithm for ECDSA* signatures. Section 3.3 is about the security analysis of our scheme, and Section 3.4 shows the comparative results and discussion followed by the summary in Section 3.5.

3.1 DEFINITIONS AND NOTATIONS

Koblitz (1987) and Miller (1985) independently introduced Elliptic Curve Cryptography, which is currently one of the most secure public key cryptosystems. ECDSA is based on the Discrete Logarithm Problem (DLP). All the values used for computation belong to the Elliptic Curve group.

Table 3.1: Notations followed in chapter

Symbol	Reference to
q	Order of the prime field $\mathbb{F}_q$.
$E(\mathbb{F}_q)$	Elliptic Curve defined over prime field $\mathbb{F}_q$.
n	Order of P, usually a prime.
P	Random non-zero base point of order n in $E(\mathbb{F}_q)$.
h	The cofactor $\frac{E(\mathbb{F}_q)}{n}$.
$H(m)$	Hash value of the message to be verified.
t	Size of the batch of signatures.

Table 3.1 shows the notations followed in the chapter for further references.

We provide generic algorithms for digital signature. The digital signature generation and verification consists of three algorithms and can be formally defined as

A **Digital Signature Algorithm** is actually a systematic study of three probabilistic Algorithms *(Gen, Sign, Vrfy)*(Katz and Lindell 2014):

- *Gen* is the Key Generation algorithm, which takes security parameter 1^l as input

and generates the *(pk, sk)* as output, where *pk* is public key and *sk* is private key. We assume that *pk* and *sk* have length at least l, and that l can be determined from *pk* and *sk*.

- *Sign* is the Signing algorithm that takes the private key *sk* and the message m as inputs and outputs signature s, which can be written as $s \leftarrow Sign_{sk}(m)$.

- *Vrfy* is the Verification algorithm, which takes the public key *pk*, message m and the signature s as inputs and outputs b whose value is either '1', if the signature is valid or '0', if the signature is invalid. It can be shown as $b \leftarrow Vrfy_{pk}(m, s)$.

Similarly the batch verification algorithm can be formally stated as:
Suppose *(Gen, Sign, Vrfy)* is a Digital Signature Scheme with 1^l as the security parameter, $k, n \in poly(l)$, $PK = pk_1, \ldots, pk_k$ and $(pk_1, sk_1), \ldots, (pk_k, sk_k)$ are generated by *Gen*(1^l), the Batch Verification Algorithm (Bellare et al. 1998) should hold the following conditions:

- If $pk_i \in PK$ and $Vrfy_{pk_i}(m_i, s_i) = 1$ for $1 \leq i \leq n$ then $Batch((pk_1, m_1, s_1), \ldots, (pk_n, m_n, s_n)) = 1$

- If $pk_i \in PK$ for all $1 \leq i \leq n$ and $Vrfy_{pk_i}(m_i, s_i) = 0$ for some $1 \leq i \leq n$, then $Batch((pk_1, m_1, s_1), \ldots, (pk_n, m_n, s_n)) = 0$ except with negligible probability in l, over the randomness of *Batch*.

The batch verification is as secure as individual verification provided the batch verification satisfies the basic digital signature properties: Authenticity, Integrity, and Non-Repudiation.

There are certain assumptions we have made for further processing. The cofactor is assumed as $h = 1$, which makes $E(\mathbb{F}_q)$ a cyclic group and P is the generator of $E(\mathbb{F}_q)$. According to Hasse's theorem, $|n - q - 1| \leq 2\sqrt{q}$. If $n \geq q$, then $\mathbb{Z}_n$ has a unique representation in $\mathbb{Z}_q$. And if $n < q$, then an element in $\mathbb{Z}_n$ has up-to two representation in $\mathbb{Z}_q$.

3.1.1 The ECDSA algorithm

ECDSA is one of the most secure and lightweight digital signature algorithm. The elliptic curve parameters are carefully designed so that attacker should not be able to solve the ECDLP in less than exponential time. In this subsection, we provide the ECDSA algorithms for key generation, signature generation and signature verification. The ECDSA domain parameters are $\{q, E(\mathbb{F}_q), P, n, h\}$. These parameters remain the same for ECDSA and ECDSA* signatures. These parameters are briefed in Table 3.1.

Algorithm 3.1: ECDSA Key Generation Algorithm

Input: Standard domain parameters
Output: Key Pairs: Public Key- Q, Private Key- d

1. For an elliptic curve $E(\mathbb{F}_q)$, choose P of order n, $P \in E(\mathbb{F}_q)$
2. Choose a random integer d such that $2 \leq d \leq n-2$.
3. Compute $Q = dP$.

The Algorithm 3.1 is for key generation, where the public key and private key for ECDSA signatures are generated using domain parameters. The Algorithm 3.2 refers to signature generation. The key generation and signature generation phase of both ECDSA and ECDSA* signatures are same.

Algorithm 3.2: ECDSA Signature Generation Algorithm

Input: Message to be signed m, Private Key d, Elliptic Curve Domain Parameters
Output: ECDSA Signature (m, r, s)

1. Signer chooses a random integer k, such that $2 \leq k \leq n-2$
2. Compute $R = kP$
3. $r = x(R) (\text{mod } n)$, where $x(R)$ is x- coordinate of R.
4. Compute $s = k^{-1}(h(m) + dr) \text{ (mod } n)$

The Algorithm 3.3 is for verification of ECDSA signatures. The algorithm briefs the steps for verification of individual ECDSA signature.

Algorithm 3.3: ECDSA Signature Verification Algorithm

Input: ECDSA signature (m, r, s), Public Key Q
Output: Signature Accept or Reject

1. Verifies if $(r, s) \in [1, n-1]$, else rejects the signature.
2. The verifier computes $w = s^{-1} (\text{mod } n)$
3. Compute $u = h(m)w \text{ (mod } n)$
4. Compute $v = rw \text{ (mod } n)$
5. Calculate value of $R.y$ to retrieve point R through square root method from received r value
6. Compute $R = uP + vQ \in E(\mathbb{F}_q)$, and accept the signature if and only if $x(R) = r \text{ (mod } n)$, where $x(R)$ is x- coordinate of R.

3.1.2 The ECDSA* algorithm

Next, we provide the algorithms for key generation, signature generation, and signature verification of ECDSA* signatures. The ECDSA* key generation and signature generation algorithms are the same as for ECDSA signatures. The only difference is in the signature size and the signature verification algorithm. The signature of ECDSA* signature scheme is (R, s) whereas the original ECDSA signature is (r, s) with respect to (m, Q), where $r = f(R)$, m is the message to be signed and Q is the public key. The key generation and signature generation algorithms are provided in Algorithms 3.4 and 3.5 respectively.

Algorithm 3.4: ECDSA* Key Generation Algorithm

Input: Standard domain parameters
Output: Key Pairs: Public Key- Q, Private Key- d

1. For an elliptic curve $E(\mathbb{F}_q)$, choose Base point P of order n, $P \in E(\mathbb{F}_q)$.
2. Choose a random integer d such that $2 \leq d \leq n-2$.
3. Compute $Q = dP$.

The Algorithm 3.6 is the verification algorithm for only one ECDSA* signature. The naive batch verification equations for ECDSA* signatures are same as the naive

Algorithm 3.5: ECDSA* Signature Generation Algorithm

Input: Message to be signed m, Private Key d, Elliptic Curve Domain Parameters

Output: ECDSA* Signature (m, R, s)

1. Signer chooses a random integer k, such that $2 \leq k \leq n-2$.
2. Compute $R = kP$.
3. $r = x(R) (\text{mod } n)$ \\$x(R)$ is x- coordinate of R.
4. Compute $s = k^{-1}(H(m) + dr) \text{ mod } n$.

Algorithm 3.6: ECDSA* Signature Verification Algorithm

Input: ECDSA* signature (m, R, s), Public Key Q

Output: Signature Accept or Reject

1. Verifies if $s \in [1, n-1]$, else rejects the signature.
2. The verifier first computes r, $r = x(R)$.
3. The verifier computes $w = s^{-1} \text{ mod } n$.
4. Then compute $u = H(m)w \text{ mod } n$.
5. Compute $v = rw \text{ mod } n$.
6. Compute $R = uP + vQ$, and accept the signature if and only if $x(R) = r \text{ mod } n$ \\$x(R)$ is x- coordinate of R.

batch verification equations for ECDSA signatures as provided in Equations 3.1 and 3.2. The difference between the two lies in the initial verification, where extra computation of y_i-coordinate of point R_i in ECDSA is needed, which is not needed in ECDSA*. The other difference is the signature size. Since ECDSA* sends entire point R_i, the y_i-coordinate value is also included in the signature along with x_i-coordinate value. Hence ECDSA* signature size is little higher than ECDSA signature. For a batch of ECDSA* signatures or ECDSA, the last step of naive batch verification test is same and has two cases. The first case is when the signatures in the batch are generated by the single signer and the equation for the same is Equation 3.1, and the second case is when the signatures are generated by multiple signers denoted by Equation 3.2.

In case of single signer,

$$\sum_{i=1}^{t} R_i = \left(\sum_{i=1}^{t} u_i\right) P + \left(\sum_{i=1}^{t} v_i\right) Q \tag{3.1}$$

In case of multiple signers,

$$\sum_{i=1}^{t} R_i = \left(\sum_{i=1}^{t} u_i\right) P + \sum_{i=1}^{t} v_i Q_i \tag{3.2}$$

In the case of multiple signers, the efficiency decreases. As the number of scalar multiplications increases, the execution time increases too.

3.2 PROPOSED BATCH VERIFICATION SCHEME

The nodes in the IoT network are more vulnerable to various security attacks because of their low computation capability and memory because of which security protocols are difficult to implement. They also have low battery capacity, which needs to be utilized efficiently so that the routine tasks of the nodes are not affected. The existing batch verification schemes are efficient for signatures of both single and multiple signers. Although the existing batch verification schemes are secure and efficient, the efficiency ceases as the size of the batch increases. In real time scenario, the batch size is relatively higher, and the existing schemes will not be suitable under such conditions.
The proposed scheme performs well for various batch sizes at the price of a few integer multiplication operations and also provides more security.

When multiple ECDSA* signatures generated by either single signer or multiple signers, are received, the verifier performs batch verification to verify these signatures at once. Our scheme is derived from the concept introduced by Naccache et al. (1994) and is a variation of the scheme introduced in Karati et al. (2012a). In our scheme, the verifier generates the pairwise relative prime equation randomizers $b_1, b_2, \ldots, b_t < n, \in \mathbb{F}_q$ during verification. The generation of the pairwise random numbers will incur a little delay in verification, but it is very insignificant as compared to the advantage in the gain of efficiency and security.

The pairwise relative primes are the ones where any pair of numbers are relatively prime with each other, i.e., the GCD of any pair of numbers is one. The relative prime

numbers $b_1, b_2, \ldots, b_t$ can be generated by considering a set M of natural primes which are less than n. To optimize the cardinality of M, let us denote one more set C of first g primes. Let $f(g)$ be such that $S_g^{f(g)}$(Sterling numbers of second kind) is maximal. Sterling numbers of the second kind is a concept in combinatorics, which indicates the number of ways in which a set of g numbers can be partitioned into $f(g)$ empty subsets. For computing $b_1, b_2, \ldots, b_{f(g)}$ during verification, the set of primes C has to be partitioned into $f(C)$ classes $C_1, C_2, \ldots, C_{f(C)}$, and the elements in each C_i are inter-multiplied to obtain b_i. Hence to generate $f(g)$ pairwise random numbers, we just need g primes, where $(f(g) > g)$. Hence we optimized the size of set C. Therefore if we had generated only primes instead of pairwise primes, then the cardinality of C would have been very large, and it would become difficult for the verifier to maintain such a large set. Hence to reduce memory usage as well as to improve security, we used pairwise random numbers.

This can be explained with an example as follows: Consider the set of first 6 primes $C = \{2, 3, 5, 7, 11, 13\}$. To find relative primes, create three subsets and assign the primes randomly from set C to these subsets, $C_1 = \{2, 11\}$ and $C_2 = \{3, 13\}$ and $C_3 = \{5, 7\}$. Now by inter-multiplying any number of elements in C_i, we obtain b_i. Therefore $b_1 = \{2 \times 11\}$, $b_2 = \{3 \times 13\}$, $b_3 = \{5 \times 7\}$ and so on. Hence if only three b's are needed, then $b_1 = 22$, $b_2 = 39$ and $b_3 = 35$. The three numbers $\{b_1, b_2, b_3\}$ = $\{22, 39, 35\}$ are relatively prime with each other. If we pick any two numbers of the three, their GCD will be one. For our scheme we are generating the co-primes of size 32-bit. For an intruder, it is challenging to guess these numbers which are generated randomly during verification.

For the given standard elliptic curve parameters $[a, b, q, P, n]$, our scheme generates pairwise relative prime randomizers in the range $[1 \ldots n]$. The reason for implementing our scheme on ECDSA* is because the verification of ECDSA* signatures takes less time compared to ECDSA signatures. The y-coordinates of the point R_i at the LHS of Equations 3.1 and 3.2 in ECDSA are unknown when the verifier receives signatures. Hence ECDSA requires extra time for verification since the computation of the y-coordinates for each R_i involves solving for modular square roots. But in ECDSA*,

y-coordinates of the point R_i are known since the curve point R is sent directly in the signature to the verifier. Hence the time to compute y-coordinate is reduced in ECDSA* signatures.

Algorithm 3.7: Our_Scheme Key Generation Algorithm

Input: Standard domain parameters
Output: Key Pairs: Public Key- Q, Private Key- d

1. For an elliptic curve $E(\mathbb{F}_q)$, choose P of order n, $P \in E(\mathbb{F}_q)$.
2. Choose a random integer d such that $2 \leq d \leq n-2$.
3. Compute $Q = dP$.

Algorithm 3.8: Our_Scheme Signature Generation Algorithm

Input: Message to be signed m, Private Key d, Elliptic Curve Domain Parameters
Output: ECDSA Signature (m, R, s)

1. Signer chooses a random integer k, such that $2 \leq k \leq n-2$.
2. Compute $R = kP$.
3. $r = x(R) \bmod n$, where $x(R)$ is x- coordinate of R.
4. Compute $s = k^{-1}(H(m) + dr) \bmod n$.

The verification equation in case of single signer is given in Equation 3.3,

$$\sum_{i=1}^{t} R_i b_i = \left(\sum_{i=1}^{t} (b_i u_i) (\bmod n) \right) P + \left(\sum_{i=1}^{t} (b_i v_i) (\bmod n) \right) Q \tag{3.3}$$

The verification equation for multiple signers is given in Equation 3.4

$$\sum_{i=1}^{t} R_i b_i = \left(\sum_{i=1}^{t} (b_i u_i) (\bmod n) \right) P + \left(\sum_{i=1}^{t} (b_i v_i) (\bmod n) Q_i \right) \tag{3.4}$$

Our scheme is implemented for ECDSA* signatures to reduce the overhead of calculating $R.y$. Our proposed scheme can also be implemented for ECDSA signatures too. The three Algorithms (3.7, 3.8, 3.9) explain our scheme for single and multiple signers. In the verification Algorithm 3.9, b_i's are the relative prime numbers whose generation has already been explained. The scheme incurs additional integer multiplication for every signature. But this increased computation cost does not

Algorithm 3.9: Our_Scheme Signature Verification Algorithm

Input: t number of ECDSA* signatures (m_i, R_i, s_i), Public Key Q
Output: Signature Accept or Reject

1. Chooses t pairwise relatively prime randomizers $b_1, b_2, \ldots, b_t < n, \in \mathbb{F}_q$.
2. The verifier first computes r_i, $r_i = x(R_i)$, where $x(R)$ is x- coordinate of R.
3. The verifier computes $w_i = s_i^{-1} \text{mod } n$.
4. Compute $u_i = h(m_i)w_i \text{ mod } n$.
5. Compute $v_i = r_i w_i \text{ mod } n$.
6. Compute the LHS and RHS separately and verify the signatures with either of Equations 3.3 or 3.4.

significantly affect the performance of the verification algorithm. The b_i is generated for every signature to be verified. Hence the intruder cannot guess the value of b_i beforehand.

Hence with ECDSA* signatures, the overhead of computing point R by calculating $R.y$ is reduced at the verifier. The signer sends the entire point R in the signature along with s, instead of $R.x$ and s. Therefore one major step of calculating $R.y$ is reduced during verification. In the results section, we provide the performance of our scheme implemented for ECDSA* signatures.

As per the results in Antipa et al. (2005), implementing ECDSA* signatures increases the performance of verification by 40%. Therefore from the algorithm for ECDSA* signature verification, we can observe that the amount of time spent in calculating the $R.y$ for every signature is reduced with a minor increase in the signature size. The proposed scheme is very similar to the scheme introduced by Karati et al. (2012a). The only difference is the generation of b_i during verification. In the Karati et al. (2012a) scheme, the b_is are generated randomly, and in the proposed scheme, we are using random relative prime numbers which require less memory space and provide more security.

Our focus is on developing a batch signature verification scheme for IoT. The IoT

environment requires a fast and secure batch verification scheme since they have low computing power, which makes them more vulnerable to external attacks. Hence a scheme which requires less computation and is more secure proves to be suitable for IoT. Therefore the proposed scheme, which is faster as well as secure compared to most of the other batch verification schemes, is more suitable for IoT applications.

3.3 SECURITY ANALYSIS

In ECDSA* algorithms, the signature (m_i, R_i, s_i) is generated by the signer. The proposed scheme is a batch verification scheme for ECDSA* signatures. Our scheme is different from naive batch verification scheme for ECDSA* signatures, only in verification algorithm.

For ECDSA* signature scheme, the entire R_i is part of the signature generated by the signer, which reduces the time of reconstruction of y_i at the verifier. But this exposes entire R_i to the adversary. Hence disclosing $R_1, R_2, \ldots R_t$ will expose both x_i and y_i coordinates of every R_i, which makes the naive batch verification scheme for ECDSA* signatures vulnerable to forgery attacks. Hence introducing t pairwise relative prime randomizers makes such forgery attacks infeasible. The attacker finds it very difficult to guess the numbers generated. A l-bit relative prime number can enhance security by 2^l. We have also presented a forgery attack later on the schemes which use only random prime numbers during batch verification. Even though the proposed batch verification scheme experiments for ECDSA* signatures, the scheme can also be applied to ECDSA signatures too, since both the schemes are equally secure (Antipa et al. (2005)).

3.3.1 Possible Attacks on the Existing Schemes

1. In the attack stated by Bernstein et al. (2012), an attacker generates two forged signatures among the t signatures. Therefore the verifier has only $t-2$ valid signatures. If the two forged signatures are (r, s) and $(r, -s)$ for the same message m, generated with the same key pair (d, Q), then the naive batch verification algorithm verifies and accepts all the t signatures. This attack on naive ECDSA and ECDSA* is successful when all the other signatures are from the same signer.

To overcome such an attack, our scheme generates random pairwise primes during

verification. Therefore it becomes challenging for the attacker to guess the number generated at the verifier. For a l bit relative prime, it takes around 2^l tests to arrive at the number, which is practically impossible to guess for the attacker. Hence our scheme can tackle this attack since the attacker cannot guess the random number generated by the verifier during verification.

2. In the second attack as stated in Karati et al. (2012a), the attacker has knowledge of the valid key pair (d_1, Q_1). The attacker tries to forge the signature for message m_2 by generating (r_2, s_2) under any valid key Q_2 along with the key Q_1 for message m_1. The values r_2 and s_2 can be found as, $R_2 = k_2P$ and $r_2 = x(R_2)$ with random s_2. Now for another message m_1, R_1 is computed as $R_1 = r_2 s_2^{-1} Q_2$ and $r_1 = x(R_1)$ and $s_1 = (H(m_1) + r_1 d_1)(k_2 - H(m_2)s_2^{-1})^{-1}$. During batch verification using naive method, $R_1 + R_2 = (H(m_1)s_1^{-1} + H(m_2)s_2^{-1})P + r_1 s_1^{-1} Q_1 + r_2 s_2^{-1} Q_2$, which is same as $(k_2 P + r_2 s_2^{-1} Q_2)$. Therefore these signatures are verified as valid for the naive batch verification.

In this attack, the attacker tries to forge the signatures and pass the verification test. But he/she fails to guess the relative prime number generated during verification for every signature. Hence the forger has to guess this number of l bits length from 2^l possibilities for every signature which is practically impossible. In our scheme, we are generating 32-bit co-prime numbers at the verifier, which challenges the attacker to try all 2^{32} possibilities to arrive at the number. Therefore our verification scheme ceases the possibility of accepting these invalid signatures.

3. In the third attack, we explain the possibility of an attack if we use the random prime numbers instead of relative primes for verifying ECDSA* signatures.

Theorem 3.1
The following statements are equivalent:

(a) There is an efficient batch verification algorithm
$Batch(m_1, m_2, b, b_1, b_2, P, Q) = (R_1, R_2, s_1, s_2)$ *for verifying ECDSA* signatures (where m_1, m_2 are the messages and b, b_1, b_2 are randomly generated primes) such*

that:

$$R_1 b + R_2 b_i \\ = (b s_1^{-1} H(m_1) + b_i s_2^{-1} H(m_2)) \bmod n\ P + (b s_1^{-1} r_1 + b_i s_2^{-1} r_2) \bmod n\ Q \quad (3.5)$$

where $b \neq b_1 \neq b_2$

(b) An algorithm exists to break this batch verification test.

Proof: In order to prove that $(a) \implies (b)$, one has to choose b, b_1, b_2 in such a way that $b' = b_1 - b_2$ has an inverse modulo n and compute $Batch(m_1, m_2, b, b_1, b_2, p, Q) = (r_1, r_2, s_1, s_2)$

$$R_1 b + R_2 b_i = (b s_1^{-1} H(m_1) + b_i s_2^{-1} H(m_2)) \bmod n\ P + (b s_1^{-1} r_1 + b_i s_2^{-1} r_2) \bmod n\ Q \quad (3.6)$$

Now by subtracting the Equation 3.6 for $i = 1, 2$, we get

$$(R_1 b + R_2 b_1) - (R_1 b + R_2 b_2) = \\ ((b s_1^{-1} H(m_1) + b_1 s_2^{-1} H(m_2)) \bmod n\ P + (b s_1^{-1} r_1 + b_1 s_2^{-1} r_2) \bmod n\ Q)\ - \\ ((b s_1^{-1} H(m_1) + b_2 s_2^{-1} H(m_2)) \bmod n\ P + (b s_1^{-1} r_1 + b_2 s_2^{-1} r_2) \bmod n\ Q) \quad (3.7)$$

Therefore the equation leads to,

$$R_2 b' = (b' s_2^{-1} H(m_2)) \bmod n\ P + (b' s_2^{-1} r_2) \bmod n\ Q \quad (3.8)$$

Now by taking b'^{th} roots mod n on both the sides, since b' has an inverse modulo n, the signature (m_2, r_2, s_2) passes the batch verification and individual ECDSA* verification tests.

Therefore a lemma can be stated as the immediate consequence to Theorem 3.1:

Lemma 3.2
If all the b_is are picked in a set M and are pairwise relatively prime, and are smaller than n then,

$$\Omega(M) = Pr\big[BATCH(b_i \in M, \{\{r, s\}, \{r', s'\}\}) = True| \\ ECDSA^*(\{r, s\}) \wedge ECDSA^*(\{r', s'\}) = False\big] \\ = \tfrac{1}{|M|^2 - |M|} \cong |M|^{-2} \quad (3.9)$$

Proof: Here the probability of batch verification returning $True$ and two ECDSA*

verification having the same b is inversely proportional to the cardinality of M. To prove the same, consider a natural set M of primes which are less than certain bound, which is decided depending upon the application. To optimize the cardinality of M, lets us denote M as a set of first m primes and let $f(m)$ be such that $S_m^{f(m)}$ is maximal. For computing $b_1, b_2, \ldots, b_{f(m)}$, M has to be randomly partitioned into $f(m)$ classes $M_1, M_2, \ldots, M_{f(m)}$, and the elements in each M_i are multiplied in combination within each class, to obtain b_i. This generates pairwise prime numbers from an optimized set M.

From the Lemma 3.2, it is clear that we can reduce the cardinality of M and still manage to generate relative co-prime numbers more than the cardinality of M. That is, if the cardinality of M is m, then we can still generate more than m pairwise prime numbers. That is the reason behind considering pairwise co-prime random numbers rather than random primes. If only primes are considered, then the size of M has to be very large. Hence we are generating pairwise co-primes, to restrict the cardinality of M and generate more b's. These co-primes have an added advantage in security. If the numbers are only prime numbers, then more than 50% of the computation for the attacker is reduced, since all the even and composite numbers are eliminated.

3.4 RESULTS AND ANALYSIS

In this section, we provide the results of implementing the proposed batch verification scheme for ECDSA* signatures. We also provide the results of existing batch verification schemes for ECDSA and make a comparative study. All the parameters considered for experimentation are standard parameters suggested by NIST (National Institute of Standards and Technology).

We have applied the proposed batch verification scheme on three NIST suggested curves: *(P-192)*, *(P-224)* and *(P-256)* curves. We have analysed the runtime values, the speedup values of all the three curves. We have also provided the speedup and verification time values for multiple signers as well single singer, for all the curves.

The experimentation is carried out on a Rock cluster CentOS 6.0 system. The processor is Intel® Xeon® E5-2650. There are seven machines, and each machine has

twenty cores. And each core runs on 2.3 GHz processor. We have implemented using the library CyptoPP. CryptoPP or Crypto++ is an open source class library for C++ for cryptographic algorithms and functions. CryptoPP supports most of the cryptographic schemes and primitives. The library supports a variety of compilers and all major operating systems.

We have compared the verification time and speedup values of various schemes in this section. The columns indicate the verification time values for different schemes, including our schemes.

- Batch Size: Indicates the number of signatures in the batch.
- Individual: These values indicate the time of verification if the signatures are verified individually.
- Naive_ECDSA: This indicates the time taken by the naive batch verification scheme for ECDSA signatures.
- Naive_ECDSA*: It indicates the verification time taken by the naive batch verification scheme for ECDSA* signatures.
- S1: The verification time for the Algorithm S1 for ECDSA signatures.
- S2: The verification time for the Algorithm S2 for ECDSA signatures.
- Our_Scheme: The verification time taken by the proposed batch verification scheme for ECDSA* signatures.

3.4.1 Verification Times

The proposed batch verification scheme for ECDSA* signatures is compared with existing batch verification schemes for ECDSA signatures. The results are also compared with the sequential verification of ECDSA signatures. We first provide the verification time values for single and multiple signers and later we discuss the speedup gained by using the batch verification schemes over sequential verification.

Single Signer

For single signer, we have provided results for all the three curves. We can observe one interesting fact that, the curve *P-192* takes more verification time compared to the other two curves. The performance of the curve *P-224* and *P-254* are almost same. Among the two, the curve *P-224* proves to be mostly suited for batch verification schemes for ECDSA signatures as well as individual verification for single signer.

Table 3.2: Verification Time for the curve ***(P-192)*** for Single Signer (sec)

Batch Size	Individual	Naive ECDSA	Naive ECDSA*	S1	S2	Our scheme
2	0.108	0.056	0.056	0.056	0.055	0.057
3	0.498	0.184	0.175	0.191	0.18	0.182
4	0.709	0.214	0.184	0.275	0.214	0.198
5	0.896	0.256	0.194	0.605	0.297	0.25
6	1.163	0.298	0.213	2.326	0.546	0.302
7	1.626	0.351	0.258	6.775	4.065	0.354
8	2.213	0.427	0.313	-	-	0.517
16	9.201	0.892	0.661	-	-	0.963
32	28.64	1.429	1.108	-	-	1.844
64	78.32	2.728	2.255	-	-	3.341

Table 3.3: Verification Time for the curve ***(P-224)*** for Single Signer (sec)

Batch Size	Individual	Naive ECDSA	Naive ECDSA*	S1	S2	Our scheme
2	0.1575	0.091	0.090	0.081	0.08	0.091
3	0.280	0.118	0.113	0.104	0.113	0.098
4	0.412	0.144	0.135	0.146	0.120	0.140
5	0.536	0.180	0.145	0.319	0.161	0.151
6	0.732	0.242	0.178	1.22	0.293	0.193
7	0.888	0.256	0.191	4.44	0.986	0.203
8	1.067	0.273	0.212	-	-	0.235
16	2.74	0.403	0.333	-	-	0.421
32	8.94	0.789	0.663	-	-	0.803
64	32.11	1.5408	1.284	-	-	1.596

Tables 3.2, 3.3 and 3.4 represent the verification time taken by the three curves for a batch of signatures signed by single signer. We can observe from the tables that, the verification time taken by the naive batch verification schemes is less than the proposed

Table 3.4: Verification Time for the curve ***(P-256)*** for Single Signer (sec)

Batch Size	Individual	Naive ECDSA	Naive ECDSA*	S1	S2	Our scheme
2	0.153	0.079	0.079	0.078	0.077	0.079
3	0.276	0.102	0.096	0.105	0.097	0.102
4	0.457	0.135	0.12	0.154	0.13	0.131
5	0.663	0.173	0.14	0.338	0.19	0.157
6	0.866	0.240	0.162	1.209	0.307	0.187
7	1.093	0.225	0.172	2.876	1.271	0.201
8	1.385	0.238	0.186	-	-	0.223
16	4.737	0.408	0.332	-	-	0.431
32	18.703	0.758	0.627	-	-	0.815
64	76.462	1.52	1.275	-	-	1.630

scheme, since our scheme generates the relative pairwise primes for security. Therefore our scheme has higher number of modular addition and scalar multiplication operations. We have already discussed the attacks on the naive verification. S1 and S2 schemes are efficient for smaller batch sizes, but as the batch size increases, they become impractical. Hence they are not applied for higher batch sizes. The batch verification schemes definitely have significant improvement over individual verification of signatures in sequential manner.

Multiple Signers

In case of verification time for multiple signers, again the curve *P-224* performs better than the other two curves. In case of multiple signers verification time, the batch of signatures contains signatures signed by more than one signer. Hence such a batch requires more time for verification. The verification time values for individual verification does not change for single and multiple signers.

Tables 3.5, 3.6 and 3.7 represent the verification time values for batch of signatures signed by multiple signers. Table 3.5 is for the curve *P-192* and the Tables 3.6 and 3.7 are respectively for the *P-224* and *P-256* curves. The efficiency gained through batch verification for multiple signers over individual verification is not very significant as compared to the efficiency gained for single signers. In case multiple signers too, the proposed scheme's performance is comparable to the performance of naive batch

Table 3.5: Verification Time for the curve ***(P-192)*** for Multiple Signers (sec)

Batch Size	Individual	Naive ECDSA	Naive ECDSA*	S1	S2	Our scheme
2	0.108	0.08	0.083	0.087	0.08	0.09
3	0.498	0.35	0.339	0.356	0.35	0.389
4	0.709	0.477	0.472	0.545	0.51	0.537
5	0.896	0.618	0.56	0.985	0.64	0.66
6	1.163	0.868	0.705	2.91	0.99	0.843
7	1.626	1.355	0.945	5.42	1.827	1.169
8	2.213	1.75	1.28	-	-	1.574
16	9.201	7.25	5.11	-	-	6.35
32	28.64	21.12	15.31	-	-	19.48
64	78.32	55.23	41.22	-	-	52.8

Table 3.6: Verification Time for the curve ***(P-224)*** for Multiple Signers (sec)

Batch Size	Individual	Naive ECDSA	Naive ECDSA*	S1	S2	Our scheme
2	0.1575	0.13	0.127	0.122	0.12	0.135
3	0.280	0.21	0.210	0.2	0.195	0.221
4	0.412	0.307	0.290	0.31	0.275	0.311
5	0.536	0.412	0.37	0.531	0.383	0.398
6	0.732	0.567	0.47	1.49	0.653	0.536
7	0.888	0.632	0.562	2.69	1.03	0.645
8	1.067	0.751	0.667	-	-	0.768
16	2.74	1.879	1.62	-	-	1.925
32	8.94	5.87	5.02	-	-	6.144
64	32.11	20.73	17.54	-	-	21.81

verification scheme for ECDSA signatures. But the proposed scheme can not surpass the performance of naive batch verification ECDSA* signatures, because our scheme has more number of modular addition and scalar multiplication operations.

3.4.2 Speedup Values

In the previous subsection, we analysed the verification time for different NIST elliptic curves for both single as well as multiple signers. Next is to study the speedup gained by the batch verification schemes compared to ideal speedup for all the three curves. Initially we provide speedup values for batches signed by single signer. Later we discuss the speedup gained when multiple signers sign the signatures in the batch.

Table 3.7: Verification Time for the curve ***(P-256)*** for Multiple Signers (sec)

Batch Size	Individual	Naive ECDSA	Naive ECDSA*	S1	S2	Our scheme
2	0.153	0.116	0.116	0.116	0.115	0.128
3	0.276	0.194	0.19	0.194	0.189	0.215
4	0.457	0.30	0.295	0.319	0.305	0.344
5	0.663	0.44	0.404	0.619	0.44	0.487
6	0.866	0.61	0.52	1.574	0.66	0.627
7	1.093	0.761	0.63	3.036	1.23	0.786
8	1.385	0.969	0.819	-	-	0.988
16	4.737	3.025	2.513	-	-	3.273
32	18.703	11.82	10.33	-	-	12.80
64	76.462	49.04	40.456	-	-	51.63

Single Signer Speedup

In a batch of signatures signed by single signer, the ideal speedup is the expected speedup. All the other batch verification schemes are compared with the ideal speedup and the schemes whose speedup values are close to ideal speedup are considered efficient.

Table 3.8: Speedup for the curve ***(P-192)*** for Single Signer

Batch Size	Ideal	Naive ECDSA	Naive ECDSA*	S1	S2	Our scheme
2	2	1.92	1.92	1.92	1.96	1.89
3	3	2.70	2.84	2.6	2.76	2.73
4	4	3.3	3.85	2.57	3.31	3.58
5	5	3.5	4.61	1.48	3.01	3.5
6	6	3.9	5.45	0.50	2.13	3.85
7	7	4.62	6.3	0.24	0.4	4.5
8	8	5.18	7.06	-	-	4.28
16	16	10.3	13.9	-	-	9.55
32	32	20.04	25.8	-	-	15.53
64	64	28.7	34.73	-	-	23.44

The Tables 3.8, 3.9 and 3.10 represent the speedup details with respect to ideal speedup, for various batch verification schemes. For a batch size (< 6), the Algorithm S2 performs considerably well. For batch sizes higher than that, the performance of

Table 3.9: Speedup for the curve ***(P-224)*** for Single Signer

Batch Size	Ideal	Naive ECDSA	Naive ECDSA*	S1	S2	Our scheme
2	2	1.73	1.75	1.95	1.98	1.73
3	3	2.37	2.47	2.69	2.82	2.85
4	4	2.85	3.05	2.82	3.43	2.94
5	5	2.97	3.69	1.68	3.3	3.55
6	6	3.02	4.11	0.6	2.5	3.79
7	7	3.47	4.64	0.2	0.9	4.37
8	8	3.91	5.03	-	-	4.54
16	16	6.8	8.23	-	-	6.5
32	32	11.3	13.48	-	-	11.1
64	64	20.8	25	-	-	20.12

Table 3.10: Speedup for the curve ***(P-256)*** for Single Signer

Batch Size	Ideal	Naive ECDSA	Naive ECDSA*	S1	S2	Our scheme
2	2	1.94	1.94	1.96	1.98	1.92
3	3	2.70	2.87	2.62	2.84	2.7
4	4	3.38	3.81	2.97	3.5	3.48
5	5	3.83	4.73	1.96	3.48	4.22
6	6	3.6	5.34	0.71	2.82	4.63
7	7	4.86	6.35	0.38	0.86	5.44
8	8	5.82	7.44	-	-	6.21
16	16	11.6	14.27	-	-	10.99
32	32	24.67	29.83	-	-	22.95
64	64	50.09	59.97	-	-	46.9

Algorithm s S1 and S2 decreases. The speedup of S1 and S2 decreases drastically as the batch size increases. More the gain over individual verification time, more is the speedup. Hence the speedup for *P-256* is high and not *P-224*. The speedup of proposed scheme is comparable to the speedup gained by the naive batch verification for ECDSA.

Multiple Signers Speedup

The speedup gained for multiple signers can not be same as for single signers. Since the number of operations in case of multiple signers is more. The ideal speedup for multiple signers is given by $\frac{2t}{t+1}$, where t is the batch size.

Table 3.11: Speedup for the curve ***(P-192)*** for Multiple Signers

Batch Size	**Ideal**	**Naive ECDSA**	**Naive ECDSA***	**S1**	**S2**	**Our scheme**
2	**1.33**	1.28	1.30	1.23	1.34	1.17
3	**1.50**	1.42	1.47	1.4	1.41	1.279
4	**1.60**	1.5	1.50	1.30	1.39	1.321
5	**1.67**	1.45	1.60	0.91	1.4	1.358
6	**1.71**	1.34	1.65	0.4	1.17	1.379
7	**1.75**	1.20	1.72	0.30	0.89	1.391
8	**1.78**	1.26	1.73	-	-	1.406
16	**1.88**	1.27	1.80	-	-	1.448
32	**1.94**	1.35	1.87	-	-	1.470
64	**1.97**	1.41	1.9	-	-	1.483

Table 3.12: Speedup for the curve ***(P-224)*** for Multiple Signers

Batch Size	**Ideal**	**Naive ECDSA**	**Naive ECDSA***	**S1**	**S2**	**Our scheme**
2	**1.33**	1.21	1.24	1.29	1.32	1.16
3	**1.50**	1.34	1.33	1.40	1.43	1.267
4	**1.60**	1.34	1.42	1.33	1.50	1.322
5	**1.67**	1.30	1.45	1.01	1.4	1.346
6	**1.71**	1.29	1.55	0.49	1.13	1.365
7	**1.75**	1.4	1.58	0.33	0.86	1.377
8	**1.78**	1.42	1.6	-	-	1.388
16	**1.88**	1.45	1.69	-	-	1.423
32	**1.94**	1.52	1.78	-	-	1.455
64	**1.97**	1.55	1.83	-	-	1.472

The Tables 3.11, 3.12 and 3.13 represent the speedup values for a batch of signatures signed by multiple signers. The speedup gained by all the three curves for multiple signers are very similar. The speedup gained for multiple signers is not as significant as the speedup for single signers.

From all the verification time results and speedup results for the three curves and various number of signers, it is clear that the naive batch verification scheme for ECDSA* signatures is the most efficient one. But as mentioned in the possibility of attacks in Section 3.3.1, the scheme is not very secure. Hence our scheme is the little

Table 3.13: Speedup for the curve ***(P-256)*** for Multiple Signers

Batch Size	Ideal	Naive ECDSA	Naive ECDSA*	S1	S2	Our scheme
2	**1.33**	1.32	1.31	1.31	1.32	1.19
3	**1.50**	1.42	1.45	1.42	1.46	1.281
4	**1.60**	1.50	1.55	1.43	1.5	1.329
5	**1.67**	1.51	1.64	1.07	1.49	1.360
6	**1.71**	1.42	1.67	0.55	1.31	1.380
7	**1.75**	1.43	1.72	0.36	1.89	1.39
8	**1.78**	1.43	1.69	-	-	1.402
16	**1.88**	1.566	1.74	-	-	1.447
32	**1.94**	1.58	1.81	-	-	1.461
64	**1.97**	1.56	1.89	-	-	1.481

modification to the naive scheme for ECDSA* signatures and increases the security of the batch verification.

3.4.3 Computation Cost Analysis

In this subsection, we analyse the execution time of various ECC operations obtained during our experimentation. The verification time of the proposed batch verification scheme is very similar to the naive batch verification scheme for ECDSA* with a difference in extra computation of random co-prime numbers in the proposed scheme. We explain these computations in detail one by one.

We will analyse the various operations of naive batch verification of ECDSA* signatures and the proposed scheme for a given batch of t signatures.

Table 3.14: Various Operations of batch verification for Single Signer

Operation	**No. of Computations**	
	ECDSA*	Our_Scheme
Point Addition	t	t
Scalar Multiplication	2	t+2
Modular Multiplication	2t	4t

Batch verification algorithm for ECDSA* signatures has various time consuming operations such as Point Addition, Scalar Multiplication. These operations affect the

Table 3.15: Various Operations of batch verification for Multiple Signers

Operation	**No. of Computations**	
	ECDSA*	Our_Scheme
Point Addition	2t-1	2t-1
Scalar Multiplication	t+1	2t+1
Modular Multiplication	2t	4t

verification time of signatures and also help in analysing and comparing the efficiency of the various schemes. Tables 3.14 and 3.15 provide the number of operations required for verification using naive batch verification scheme and our proposed scheme for ECDSA* signatures. In Table 3.14, we can observe that the proposed scheme has relatively higher number of scalar and modular multiplications for single signer as compared to the naive batch versification scheme. Similarly 3.15 represents the operation-count for batch verification schemes for signatures signed by multiple signers.

Now we analyse the time taken by the two compute-intensive operations: Point Addition and Scalar Multiplication.

Table 3.16: Execution Time for expensive operations

Operation	**Time (sec)**
Point Addition	2.193×10^{-5}
Scalar Multiplication	3.24×10^{-3}

From Table 3.16, we can observe that scalar multiplication takes more computation time than point addition. Even though our scheme has a higher number of scalar multiplications, but it is resistant against forgery attacks which the naive batch verification scheme is not. But the number of point addition operations remains the same for both the schemes. Hence the overhead incurred is minimum compared to the security it provides.

3.4.4 Running Time Analysis

In the previous subsection, we analyzed the performance of the proposed scheme as well as a few existing schemes. The schemes S1 and S2 have various running time complexities. Most of the time is consumed in solving the multivariate equations to arrive at the value of $R.y$. Therefore the running time of the Algorithm S1 as stated by Karati et al. (2012b) is, $\Theta(m^3)$ where $m = 2^t$ and t is the batch size. The running time of the naive scheme involves the computation of t modular square roots in $\mathbb{F}_q$. If we assume the time for each square root computation as σ, then the complexity can be stated as $O((\sigma + m)t)$. We can observe that the Algorithm S1 can perform better than naive algorithm only if $(\sigma + m)t \gg m^3$ and it happens only if the batch size is small, i.e., if $t < 6$.

Similarly, Algorithm S2 has the running time complexity of $O(mt^2)$, which is better than $\Theta(m^3)$ for Algorithm S1. Therefore $(\sigma + m)t \gg O(mt^2)$ condition is not always true in all cases. For smaller batch sizes, Algorithm S2 performs better than naive algorithm whereas $(\sigma + m)t \gg m^3$ is true in most of the cases.

We analyzed that the running time of naive, S1, and S2 schemes depends on recreating the point R from $R.x$ and finding the value of $R.y$. In our scheme, we have implemented the algorithm for ECDSA* signatures where there is an increase in the signature size, which significantly reduces the computation of $R.y$. Hence the performance of our scheme is better than $S1$ and $S2$ for varied batch sizes and is in comparison with naive ECDSA verification. The naive scheme has overhead because of the square root calculation, and our scheme has overhead in generating the pairwise relative prime numbers during verification. Hence their speedup almost goes hand in hand. The complexity of generating these primes is $O(t * loglogt)$.

3.5 SUMMARY

This chapter proposes a new batch signature verification algorithm for ECDSA* signatures. The proposed scheme is suitable for devices with low computation power since the scheme uses ECDSA* signatures, which are 40% faster in verification compared to ECDSA signatures. We have shown that the proposed scheme performs

better since it is independent of the batch size of signatures. Hence even though few of the existing schemes perform better than the proposed scheme, they are not efficient for batch sizes greater than 6 ($t \geq 6$). The execution time and the speedup gained by various existing schemes and our scheme are also discussed in multiple cases like single and multiple signers, varied batch sizes, and also for different elliptic curves.

The proposed technique is based on the use of Sterling numbers of the second kind to generate the relative prime numbers during the signature verification. The chapter also discussed how the existing time-efficient techniques are vulnerable to attacks. Hence to design a batch verification scheme for IoT, we first developed a lightweight and secure verification scheme for IoT. This chapter introduced a novel batch verification approach to verify multiple signatures in the IoT network. The proposed scheme successfully identifies the presence of faulty signatures in the given batch of digital signatures efficiently.

The proposed scheme only tests for the presence of any faulty signatures in a given batch of ECDSA* signatures. But the scheme does not identify the location of the bad signature/s. To determine the location, we need other schemes which are efficient for IoT, i.e., which involves less computation. Hence our next objective is to design a scheme to identify the bad signatures in the batch that has failed the verification test.

Chapter 4

BAD SIGNATURE IDENTIFICATION IN BATCH VERIFICATION

4.1 INTRODUCTION

Many digital signatures can be verified at a time in a batch using various batch verification schemes. There are multiple batch verification schemes (Cheon and Yi 2007; Fiat 1989; Kittur and Pais 2017) available for various digital signature algorithms as discussed in Chapter 2. Most of the batch verification schemes accept a batch of signatures and verify whether the batch has any faulty signature/s. If all the signatures in the batch are valid, then the output of verification is *True*, and if one or more bad signatures exist in the batch, then it returns *False*.

The secure and efficient batch verification scheme for ECDSA* signatures proposed in Chapter 3 does not identify the index of bad signature. Hence there are various techniques introduced to identify the bad signature/s within a batch. The default way to find the faulty signature is sequential verification, where every signature is individually verified to find the faulty one. There are many other techniques such as Divide-and-Conquer (Pastuszak et al. 2000a; Seungwon et al. 2006), Hamming code verifier (Pastuszak et al. 2000a), etc. to identify the faulty signatures in a given batch of signatures. The existing bad signature identification schemes are either costly in computation or have restriction on the number of bad signatures in the batch. The Hamming code verifier has a restriction on the number of signatures in the batch to identify the faulty signatures. Hence it is essential to have schemes which do not pose

any such restrictions on the batch of signatures to identify the faulty signature. Since our aim is to design bad signature identification scheme for IoT applications. Hash function is one such function which is lightweight and secure and also a standard. But only hash based authentication does not satisfy the non-repudiation property, where it is difficult to prove the signer of the signature as explained in example 9.64 in Katz et al. (1996).

Hence the first scheme proposed is based on the hash function, which harnesses the advantages of hash based authentication scheme (Krawczyk et al. 1997). Hence in the proposed hash based bad signature identification scheme, the verifier performs batch verification, and if the output of batch verification is *False*, then the hash value of every signature is verified sequentially. To further reduce the bad signature identification time, we have also proposed schemes based on error control codes to verify each of the signatures in the batch to identify the faulty one. Hence the next scheme is based on Cyclic Redundancy Check (CRC) codes (Peterson and Brown 1961) which are Error-Detection codes to identify the bad signature. The other scheme is based on the Low-Density Parity-Check (LDPC) codes (Gallager 1962; MacKay and Neal 1996) which are error detection codes. LDPC codes are used for detecting and correcting data bits during transmission.

In all the proposed schemes, the signer generates a signature for the given message and encodes it using encoder (either Hash or CRC or LDPC encoder) to create the codeword by adding the check-bits to the signature. This codeword is sent to the verifier along with the message and signature. The verifier performs batch verification, and if the output of batch verification is $False$, then the signatures are individually decoded using decoders (either Hash or CRC or LDPC) to verify the signatures individually and identify the faulty ones.

The contributions of the chapter are :

- Proposed an efficient hash based bad signature identification scheme inspired from the advantages of HMAC (Krawczyk et al. 1997) authentication scheme.
- Proposed a novel efficient CRC-based bad signature identification scheme based

on the error-detection codes.

- Proposed a novel error correction code based scheme to identify the faulty signature using LDPC codes.

The chapter is organised as follows: Section 4.1 is for introduction, Section 4.2 briefs the existing bad signature identification schemes. In Section 4.3, we learn the hash based bad signature identification scheme in detail and Section 4.4 is for CRC based scheme and Section 4.5 for LDPC based scheme. The results for all the schemes are provided in Section 4.6 and the chapter is summarized in Section 4.7.

4.2 PRELIMINARIES

In this section, we are providing some of the important algorithms which are referred in the chapter later. First we study the three algorithms proposed by Bellare et al. (1998). These algorithms are also known as Generic Tests (GT) as stated by Pastuszak et al. (2000a), which can be used to refer to batch verification of signatures. The efficiency of these schemes depend on the size of the batch of signatures being verified. Hence in the rest of the chapter, we also refer to batch verification as Generic Test (GT).

The three GTs are: Random Subset Test presented in Algorithm 4.1, Small Exponents Test presented in Algorithm 4.2 and Bucket Test presented in Algorithm 4.3. In the GT Algorithms presented, g is the generator of the group $\mathbb{G}$ and $x_i \in \mathbb{Z}_q$ and $y_i \in \mathbb{G}$ for $i = 1, 2, \ldots, t$ with security parameter l. The GT Algorithm is represented as GT(m, s, t), indicating message m, signature s and batch size t.

Next, we study the algorithms used to find faulty signatures when the batch verification using GT fails. Hence there are multiple schemes to identify the faulty signature/s in the failed batch. In the sequential verification, the signatures in the batch are individually verified in sequence as shown in Algorithm 4.4, to identify the index of bad signature/s. This is also the default or the naive method of locating the bad signature.

The next approach in finding the index of the bad signature is the Divide-and-Conquer (DC) verifier (Pastuszak et al. 2000a; Seungwon et al. 2006). The entire

Algorithm 4.1: Random Subset Test

Input: Batch of t message-signature pairs $((m_1, s_1), (m_2, s_2), \ldots, (m_t, s_t))$
Output: Signatures Accept or Reject

1. For every signature s_i, $i = \{1, 2, \ldots, t\}$, select $b_i \in \{0, 1\}$ in random.
2. Select a set S of n signatures, where $n < t$, where b_is are equal to 1.
3. For all the selected $s_j, j \in 1, \ldots, n$, perform standard batch verification.
4. If the batch verification succeeds, accept the signatures, otherwise reject and repeat the batch verification test for various sub-set of signatures .

Algorithm 4.2: Small Exponents Test

Input: Batch of t message-signature pairs $((m_1, s_1), (m_2, s_2), \ldots, (m_t, s_t))$
Output: Signatures Accept or Reject

1. Generate random number $u_i \in \{0, 1\}^l$, for every signature s_i.
2. Apply the random number on both sides of naive batch verification equation. For RSA batch verification, compute,
   $x = \sum_{i=1}^{t} x_i u_i$ and $y = \prod_{i=1}^{t} y_i^{u_i}$ and verify if $g^x = y$.
3. If the verification succeeds, then accept the batch of signatures, otherwise reject.

Algorithm 4.3: Bucket Test

Input: Batch of t message-signature pairs $((m_1, s_1), (m_2, s_2), \ldots, (m_t, s_t))$, integer $v \geq 2$ and computes $V = 2^v$, where V are buckets
Output: Signatures Accept or Reject

1. For every s_i, $i \in \{1, 2, \ldots, t\}$, pick V_j from $j \in \{1, \ldots, V\}$ randomly. This means place all signatures randomly in the V buckets.
2. For every bucket V_j from $j \in \{1, \ldots, V\}$, apply standard batch signature verification algorithm on all signatures in the bucket.
3. Repeat the same batch verification test for $\lceil l/(v-1) \rceil$ times and accept the batch of signatures if all the sub- tests accept.

batch of given signatures is verified through one of the GTs. If the test fails, then the batch is divided into sub-batches which are again independently verified through the same GT previously used. Then the sub-batches which fail the GT are again sub-

Algorithm 4.4: Sequential Verifier

Input: Batch of t message-signature pairs $((m_1, s_1), (m_2, s_2), \ldots, (m_t, s_t))$
Output: Signatures Accept or Reject

1. If GT outputs $True$, then accept all the signatures and exit, otherwise go to next step.
2. For every signature s_i where $i \in \{1, 2, \ldots, t\}$ do,
   (a) Apply GT verification, GT$(m_i, s_i, 1)$.
   (b) If GT$(m_i, s_i, 1) = 1$, then the signature is valid, increment i and move to next signature.
   (c) If GT$(m_i, s_i, 1) = 0$, then the signature is invalid and add to the list L.
3. List all the signatures from the list L as invalid and exit

divided recursively to locate the bad signature. The DC verification algorithm is given in Algorithm 4.5.

Algorithm 4.5: DC Verifier

Input: Batch of t message-signature pairs $((m_1, s_1), (m_2, s_2), \ldots, (m_t, s_t))$
Output: Signatures Accept or Reject

1. $t = 1$, then verify the signature using the GT. If GT$(m, s, 1) = 1$, signature is valid, otherwise invalid and exit.
2. If GT$(m, s, t) = 1$, where $t > 1$, then all signatures are valid and exit, otherwise go to next step.
3. If the given batch has a few invalid signatures, then split the batch into α sub-batches, $\alpha_1, \alpha_2, \ldots, \alpha_\alpha$, where every sub-batch has a set of $\frac{t}{\alpha}$ signatures.
4. Now invoke the DC Verifier for every sub-batch α_j, where $j \in \{1, 2, \ldots, \alpha\}$.

There are batch verification techniques such as Hamming Code verifier (Pastuszak et al. 2000a) shown in Algorithm 4.6, which identify the invalid signature provided there is only one bad signature in the given batch of signatures. Hence the scheme performs batch verification and identifies the location of bad signature too, with a restriction on the batch size.

One more approach to finding the illegal signature/s is based on ID codes proposed

Algorithm 4.6: Hamming Verifier

Input: Batch of $t = 2^k - 1$, $k > 0$, message-signature pairs $((m_1, s_1), (m_2, s_2), \ldots, (m_t, s_t))$

Output: Index of bad signature

1. Apply GT on the received batch of signatures. If GT$(m, s, t) = 1$, accept the signatures and exit, otherwise go to next step.
2. Next is to create k sub-batches,

$$(m_i, s_i) = \{(m_j, s_j) | h_{i,j} = 1\}$$

   where $i = 1, \ldots, k$ and $h_{i,j}$ is an element the Hamming code parity matrix, for which $h_{i,j}$ is equal to 1, otherwise ignored.
3. Perform GT$(m_i, s_i, 2^k - 1) = \sigma_i$, for $i = 1, \ldots, k$, depending on the result of verification, the value of σ_i is decided.
4. If the value of $\sigma_i = 0$, then test is successful, else GT fails. The syndrome $(\sigma_1, \ldots, \sigma_k)$ indicates the bad signature positions.
5. Perform GT on the instances which are free from bad signatures. If the batch instance is accepted then return the index, else exit.

by Pastuszak et al. (2000b) shown in Algorithm 4.7. The scheme can not detect all the illegal signatures in the batch, it has the restriction on the number of bad signatures it can detect in a batch.

Algorithm 4.7: ID codes based Verifier

Input: Batch of t message-signature pairs $B^t = \{(m_i, s_i) | i = 1, \ldots, t\}$

Output: Index of n bad signatures

1. Apply GT on the received batch of signatures. If GT$(m, s, t) = 1$, accept the signatures and exit, otherwise go to next step.
2. The identification code $IC(u, n)$ identifies a maximum of n bad signatures in a collection of sub-instances $(B_1, B_2, \ldots, B_v)$ where $B_i \in B^u$.
3. The pattern $S =$ (GT$(B_1), \ldots,$GT$(B_v))$ identifies all the n faulty signatures.

The next bad signature identification schemes perform batch verification as well as identify the bad signature in the batch (Li et al. 2010; Ren et al. 2015). Two similar schemes are proposed based on the idea of placing signatures in the matrix

form and identifying the bad signature by batch verifying signatures in each row and column separately. One scheme considers two-dimensional matrix and is explained in Algorithm 4.8 and the other considers three-dimensional matrix. The two verifications schemes where the signatures are arranged in matrix fashion have certain disadvantages. If one of the rows or columns have more than one bad signatures, then the schemes fail to identify them. These two schemes are expensive, since the number of times the GT is performed is more and depends on the batch size.

Algorithm 4.8: Matrix based Verifier

Input: Batch of t, message-signature pairs $((m_1, s_1), (m_2, s_2), \ldots, (m_t, s_t))$
Output: Index of bad signature

1. The verifier generates a $M \times N$ matrix such that $M \times N \geq t$, and t random numbers r_i where $i = 1, 2, \ldots, t$ and $r_i \in \{1, 2, \ldots, t\}$.

2. Now the verifier fills the matrix positions with the signatures according to the following equation,

$$s(M, N) = \begin{cases} s(\lceil \frac{r_i}{N} \rceil, n), & \text{if } r_i \bmod M = 0 \\ s(\lceil \frac{r_i}{N} \rceil, r_i \bmod n), & \text{otherwise} \end{cases}$$

3. Now the verifier uses GT verification to verify every row and every column individually. At the row side,

$$(\prod_{i=1}^{N} s_{(M,i)})^e = \prod_{i=1}^{N} H(m_{(M,i)}) \bmod n$$

   At the column side:

$$(\prod_{i=1}^{M} s_{(i,N)})^e = \prod_{i=1}^{M} H(m_{(i,N)}) \bmod n$$

4. Find the rows and columns which fail the GT, and their overlapping location points us to the location of bad signature.

In next section we study the hash based scheme proposed to identify the bad signature. Since the existing schemes have the drawback in identifying the bad signature in the batch, we have proposed hash based bad signature identification scheme.

4.3 HASH BASED VERIFICATION SCHEME

Since the existing schemes either have restriction on the batch size or they are computationally very expensive, it is important to have a secure and efficient scheme to identify the bad signatures in a batch of signatures. Hence we have proposed a scheme based on hash function. Hash function is lightweight and secure. The hash function has an important property that it is a one-way function. Once the hash value for a string is generated, the string cannot be traced back from the hash value. Hence hashing has many advantages in cryptography and we have used the hash function to locate the bad signature. Hash-based bad signature identification scheme is similar to HMAC (Krawczyk et al. 1997). The algorithms for generating the codeword and verifying the same using hash function are provided in this section.

Algorithm 4.9: Hash based Signature Generation

Input: Secret Key sk and message $m_1, m_2, \ldots, m_t$
Output: $((m_1, s_1), (m_2, s_2), \ldots, (m_t, s_t))$and $(H_1, H_2, \ldots, H_t)$

1. Generate the message-signature pairs (m_i, s_i) for t messages, using the secret key sk of the signer for every message m_i.
2. For every message-signature pair (m_i, s_i), generate the hash values, $H_i = H(s_i)$.
3. Finally, send all the t hash values, message-signature pairs $((m_1, s_1, H_1), (m_2, s_2, H_2), \ldots, (m_t, s_t, H_t))$ across to the verifier.

As shown in Algorithm 4.9, we increase the length of the data to be sent across to the verifier in the network by a few bits by appending hash value check-bits to generate the codeword. This codeword is sent across to the verifier along with the message and the signature.

The algorithm described in Algorithm 4.10 is for hash based signature verification to identify the bad signature. We can see that the verifier performs the Generic batch verification test for the given batch instance. If the batch is found faulty, the verifier performs hash verification of individual signatures to locate the faulty one/s.

Algorithm 4.10: Hash based Signature Verification

Input: $((m_1, s_1), (m_2, s_2), \ldots, (m_t, s_t))$and $(H_1, H_2, \ldots, H_t)$
Output: Index of bad signature

1. Stopping case: If $t = 1$, then if GT$(m, s, 1) = 1$, then the signature is valid else is invalid.
2. If the instance x has signatures $t > 1$, then perform GT(m_i, s_i, t), where $i \in 1, 2, \ldots, t$ on the input x. If GT$(m_i, s_i, t) = 1$, return $True$ else go to next step.
3. Hashing step: For every received data (m_i, s_i, H_i), generate hash value for the signature s_i. $H_i' = H(s_i)$.
4. If $H_i' \stackrel{?}{=} H_i$. The signature is valid if the hash values are equal, otherwise invalid.

4.3.1 Comparative Analysis

In this section, we are making a comparative analysis of hash based verifier with the existing sequential verifier signatures and DC verifier for ECDSA*. During this comparative analysis, we assume that the number of bad signatures is unknown at the beginning of verification. To make a comparative analysis of the three algorithms (sequential, DC, Hashing), we consider their performance in best, average, and worst case scenarios. For convenience, we express the batch size t as α^k, where α and k are integers and n is the number of faulty signatures.

Best Case Analysis

In the best case scenario, we assume that there are no bad signatures in a given batch of t signatures. Then the verifier performs GT batch verification in all the three cases, i.e., sequential, DC and hashing, as the first step. The GT verifier outputs '$True'$ since there are no faulty signatures. Hence the number of GT verifications performed is just one in all three algorithms.

Average Case Analysis

There are two possibilities in the average case. We will study one by one:

1. When there is only one faulty signature, i.e., $n = 1$ out of $t = \alpha^k$ signatures in a given batch. But as previously mentioned, the presence of a single faulty

signature is unknown beforehand. The time complexity of the three algorithms can be explained as follows:

(a) In case of the sequential verifier, GT batch verification fails at the beginning due to the presence of faulty signature. The verifier then verifies every individual signature to identify the faulty one. Since the number of faulty signatures is not known, it has to look into every individual signature even after identifying one faulty signature. Therefore the number of operations are:

No. of GT: 1

No. of Individual Verifications: $t = \alpha^k$

(b) In case of DC verifier, the initial GT batch verification fails as explained earlier. So the verifier now divides the given batch into α sub-instances, and perform GT on every sub-instance. If any of the sub-instance fails the GT, again sub-divide the sub-instance and perform GT and go on till you identify the faulty signature. Now, in this case, there is only one faulty signature, and hence, only one sub-instance fails. Hence we continue to sub-divide the sub-instance until we reach the index of the faulty one.

No. of GT: $((\alpha - 1 + \frac{1}{\alpha})k + 1)$

(c) In hash based verifier, the verifier performs GT batch verification on the batch of t signatures. The test fails since there is one faulty signature in it. Then every signature is individually verified using hash verification. Hence the verifier has to perform t hash verifications.

No. of GT: 1

No. of Individual Hash Verifications: $t = \alpha^k$

2. Next we consider a scenario where n number of signatures in a given batch size of t signatures are faulty and $n < t$. The performance of the three algorithms can be given as follows:

(a) In case of the sequential verifier, the GT batch verification test fails, and since the number of faulty signatures is not known, the verifier checks all

the t signatures individually to identify the faulty ones.

No. of GT: 1

No. of Individual Verifications: $t = \alpha^k$

(b) In DC verifier, the GT verification fails since there are multiple invalid signatures. Suppose the batch instance is of size $t = \alpha^k$ and the number of faulty signatures is $n = \alpha^r$. In such a scenario the number of various operations is

No. of GT: $\alpha^{r+1}(k - r + 1) - 1$

(c) In the hash based verifier, average case occurs when 50% of the signatures in the given batch are faulty. Then each of the signature in the given batch has to be verified using the hash algorithm to identify the faulty ones. Therefore the number of operations performed to identify the faulty signatures is given as,

No. of GT: 1

No. of Individual Hash Verifications: t

Worst Case Analysis

The worst case happens when all the $t = \alpha^k$ signatures in a given batch are faulty, ie., $n = t$. In such a scenario, the performance of all the three algorithms are explained as follows:

1. In sequential verifier, after the failure of the GT test, every signature in the given batch is individually verified to identify the index of the faulty ones. The total number of operations is

   No. of GT: 1

   No. of Individual Verifications: $t = \alpha^k$

2. In DC verification, the verifier divides the given instance into α sub-instances, after the GT test fails. Then at every recursive step, the GT test fails for all the sub-instances since all the signatures are faulty. Hence the total number of operations is given as

   No. of GT: $\frac{n\alpha-1}{\alpha-1}$

3. In hash based verifier, the batch is first verified through GT, after which the batch fails the verification test. Then all the signatures in the entire batch are verified using hash verification. Hence the number of operations can be given as,
   No. of GT: $\alpha + 1$
   No. of Individual Hash Verifications: $t = \alpha^k$

From the various case analysis, it is clear that hash based verifier needs more number of hash computations than DC verifier in best and average cases, but in worst case, DC verifier requires more number of verifications than hash. But there is one important observation to be made here. Even though the number of hash verifications are high in hash based verification, the time taken for one hash verification is very less than time taken by GT. Time taken for one signature verification is same as the time taken by GT for verification of batch of signatures with size one. The next subsection briefs the verification time for GT and hash verifier.

4.3.2 Verification Time for Hash based verification

We performed hash based verification on a batch of signatures that fails the batch verification test proposed in Chapter 3. The hash based scheme outperforms the sequential verifier in total verification time needed to identify the faulty signature.

Table 4.1: Time required for various Verification operations

Operation	**Time in msec for 1 sign appx**	**Time in msec for 10 sign appx**
Signature Verification	10	150
Hash Verification	6	70

As it can be seen from the Table 4.1, the time needed to perform one hash verification is lesser than one signature verification. Hence in worst case, the hash verification scheme performs better than the other existing schemes. Hence for DC verifier, even though the number of GT are less than hash based verifier, hash verifier takes less time because the time for one hash verification is less than signature verification.

After the discussion of hash based verification scheme to identify the faulty signature, one might think of performing only hash verification to verify the sender and digital signature algorithm is unnecessary. Hash verification definitely verifies authenticity and integrity of the sender, but it does not verify the non-repudiation property. Hence it is difficult to prove that the message is indeed sent by the same sender who he claims to be. Therefore we ask the sender to generate the signature also along with codeword for every message, which is verified using batch verification before performing hash verification. In order to further reduce the time needed to identify the faulty signature, we are proposing CRC based code generation and verification in next section.

4.4 CRC BASED BAD SIGNATURE IDENTIFICATION SCHEME

The proposed bad signature identification scheme is based on Cyclic Redundancy Check (CRC) codes, which are error-detecting codes used in digital communication. The receiver in any communication uses error detection codes to detect if the received data has any occurrence of error during transmission. Before introducing our verification scheme, first, we will brief error detection and CRC encoding and decoding details in digital communication.

4.4.1 Error Detection Codes

Error detection is identifying errors caused during transmission between sender and receiver, due to noise or other impairments intentionally or accidentally. Hence the error detection schemes add extra bits to the data known as check bits. The sender generates check bits which are added to the data bits to form the codeword which is sent to the verifier. The verifier performs decoding operation to check if the received codeword has any errors.

There are various error detection codes available based on the Hamming distance (Sweeney 1991). Error detection codes generate check bits which are added to the data which aids in the detection of communication errors. Error detection codes identify the error in the received data, but not correct them unlike error correction codes. Some of the popular error detection codes are:

- Simple parity check
- Two-dimensional parity check
- Checksum
- Cyclic redundancy check

Simple Parity Check or One-dimension Parity Check:

This is one of the least expensive and most commonly used error detection code. In this scheme, a parity bit is added to the data before sending to the receiver. The parity bit is computed based on the number of 1's in the data. If the data block contains an odd number of 1's, then a parity bit of 1 is added to make it even and if there are even number of 1's, then 0 (zero) is added as the parity bit to the data to form the codeword. At the receiver, the parity bit is computed for the received data block and verified against the received parity bit. Hence the modifications to the codeword can be detected at the receiver.

Two-dimension Parity Check:

If even number of 1's are modified in the codeword, then the single parity check scheme does not recognize. Hence an improved two-dimension parity check scheme is introduced. Here the entire data block is arranged as a matrix or as a table. The parity bits are calculated for every row and every column. And the parity values for row and column are added to the data and are sent to the receiver. At the receiver, the parity bits are computed and verified against the received data.

Checksum:

In checksum error detection, the data bits are divided into fixed sized segments. The sender adds the data segments by using 1's compliment arithmetic to get the sum. This sum is complemented to generate the checksum. This checksum is sent to the receiver along with the data segments. Then at the receiver end, all the data segments are added, including the checksum using 1's compliment arithmetic. Then this sum is complemented to verify the data bits. If the result of the compliment is zero, then there are no errors in the received data or else there are errors.

Cyclic Redundancy Check (CRC):

CRC codes are one of the error detection codes, where the check bits or CRC bits are calculated through binary division and are appended to the data unit to form the codeword. The codes and other vectors are expressed as polynomial in x, where the co-efficient of the exponent of x acts as the place marker, where a polynomial of length l has a polynomial degree of $l - 1$.

The sender and receiver have a pre-agreed binary number, also known as generator polynomial, whose coefficients represent the binary number, helps to generate codeword and verify the same. The sender performs binary division of the data unit by the pre-agreed number, and the remainder is appended to the data unit and sent to the receiver. At the receiver, the codeword is divided by the pre-agreed number, and if the remainder is zero, then the data is accepted, else rejected.

To calculate the CRC codes, first, append the data of length k bits with zeros of length $l - 1$, where l is the length of the pre-agreed binary number. Therefore to get a codeword of length n bits, the data is appended with $n - k$ number of zeroes. Then divide the data by the generator polynomial. The appended zeroes to the data are replaced with the remainder of the division to form the n-bit codeword. And this codeword is sent to the verifier. The codeword for the polynomial can be given as,

$$c(x) = d(x) + r(x)$$

where $d(x)$ is the data and $r(x)$ is the remainder. A generator polynomial $g(x)$ of length $n - k$ is needed to generate the CRC bits. The CRC bits are generated as,

$$r(x) = R_{g(x)}(d(x)x^{n-k})$$

where $(R_{g(x)}(d(x)x^{n-k}))$ is the remainder obtained after dividing $d(x)$ by $g(x)$. These bits are appended to the data bits and sent. At the receiver, the received codeword is divided by the generator polynomial. If the remainder of division is zero, then there are no errors, else there are. Since we append the remainder of division to the data to generate codeword at the sender side, the codeword becomes perfectly divisible by $g(x)$, unless it has been modified during transmission. Hence we are using the concept of error detection codes in batch verification to identify faulty signatures quickly.

The CRC error detection can be explained with an example as follows: Suppose the coefficient of the generator polynomial is a 4-bit number 1011, and the data to be sent is 1101. Now during the modulo-2 arithmetic division, the data 1101 is appended with three zeroes 1101000 and is divided by 1011. The remainder is three-bit number 001, which replaces three zeroes in the data to form 1101001 codeword. This codeword is sent to the receiver. The receiver divides the codeword by the generator polynomial. If the remainder of division results in zero, then there are no errors in the data received, and if the result is non-zero, then there is a presence of error in the received data.

The drawbacks of some of the existing error detection codes are as follows:

- In simple parity checking, only single bit error can be detected. However, if two errors appear in the same codeword, then the verifier considers it as a valid codeword. Hence the simple parity check scheme can only detect an odd number of errors in the codeword.

- In two dimension parity checking, the probability of detecting the errors in the codeword increases drastically. But there is certain error which goes undetected. If two bits in the data unit are erroneous and two bits in another data unit at precisely the same position are damaged, then this error detection code cannot detect such an error.

- The checksum scheme detects all kinds of errors, including the odd bit and even bit errors too. But the drawback is that there is a possibility of two different data segments to have the same checksum, which creates ambiguity. Hence it is risky in practical implementation.

- CRC does not have any of the drawbacks as mentioned above. CRC can detect single-bit, double-bit errors; it also can detect an odd number of errors too. The chances of CRC making errors is $\frac{1}{2^c}$, where c is the length of check-bits. The error happens when more than one message has the same CRC check-bits. Hence the probability of error is $1 - \frac{1}{2^c}$ (Wolf and Blakeney 1988). Hence the reason we considered CRC for authentication.

4.4.2 CRC Verification Algorithm

In this subsection, we provide the proposed algorithms for CRC code generation at the signer and verification at the verifier designed to identify the bad signature. The signer generates the CRC check bits for every signature generated for every message as shown in Algorithm 4.11. The CRC bits are appended to the signature and sent across to the verifier (Kittur et al. 2019).

Algorithm 4.11: CRC based Signature Generation

Input: Secret Key sk and message $m_1, m_2, \ldots, m_t$, CRC generator $g(x)$
Output: $((m_1, s_1), (m_2, s_2), \ldots, (m_t, s_t))$ and CRC codewords $c_1(x), c_2(x), \ldots, c_t(x)$

1. Generate the message-signature pairs (m_i, s_i), for all the t messages using the signer's secret key.
2. For every signature s_i, calculate the CRC polynomial bits $r_i(x)$ of length $n - k$ bits using the pre-agreed binary number of bit length l bits.
3. Finally append the CRC code bits $r_i(x)$ to every signature s_i respectively to create the codeword $c_i(x)$.

The verifier receives the message-signature pairs (m_i, s_i) and the codewords for the signatures, $c_i(x)$ and performs batch verification. If the batch verification test fails, then the signatures are verified through their CRC codes, to identify the faulty signature and can be shown in Algorithm 4.12.

The algorithms in Algorithm 4.11 and 4.12 for signature generation and verification describe the steps to identify the faulty signature among a batch of signatures. Once the batch verification fails, we perform binary division of data unit $c_i(x)$ by the generator polynomial $r(x)$. If even a single bit is altered or modified, then the remainder will be non-zero, which proves that the signature is faulty.

4.4.3 Security Analysis

The security of this scheme depends on many factors. One major advantage of the scheme is that it can even identify a minor modification such as a single bit flip in the received codeword. This makes it very powerful, but this can not be completely relied

Algorithm 4.12: CRC based Signature Verification

Input: Public Key pk and t message-signature pairs $(m, s) = (m_1, s_1), (m_2, s_2), \ldots, (m_t, s_t)$, codewords $c_1(x), c_2(x), \ldots, c_t(x)$, CRC generator $g(x)$

Output: Index of the bad signature

1. Stopping case: For $t = 1$, if GT$(m, s, 1) = 1$, then the signature is valid else is invalid.
2. If $t > 1$, perform GT(m, s, t) on all the received signatures. If GT$(m, s, t) = 1$, return $True$, else go to next step.
3. Code Verify: For every signature received, $(m_i, s_i, c_i(x))$, divide the data unit $c_i(x)$ of every signature by the polynomial bits $g(x)$. If the division leaves no remainder then the signature is valid, otherwise invalid.

upon for authentication, since it just verifies the integrity of the data. Hence we perform the batch verification before performing CRC based verification. Batch verification is an authentication scheme which satisfies authentication, integrity and non-repudiation. Hence having a secure and efficient batch verification scheme is equally important. Hence the scheme proposed in Section 3.2 is used as the batch verification scheme in all our references, since we have proved it to be secure against most of the attacks.

In case of Man-in-the-Middle attack, if the attacker modifies the signature intentionally or unintentionally, the CRC verification fails. When an attacker modifies one or more bits of signature, the polynomial division ends up with a remainder, which results in a conclusion that the signature is faulty.

If an attacker modifies the signature, then the received data unit is $c'(x)$. Then the batch verification using one of the GTs fails. Then we perform CRC verification.

$$r'(x) = c'(x)/g(x)$$

$$r'(x) \neq 0$$

Since $c'(x)$ is modified, the division does not end with zero remainder, and the faulty signature can be detected.

The next attack can be the modification of CRC bits itself. But the verifier does not know the Generator Polynomial, which is shared between the signer and verifier

beforehand. Hence without the knowledge of it, it cannot generate the CRC bits. Hence the attacker does not even know the length of CRC bits and signature. Therefore the attacker does not exactly know which part of the codeword is signature and which part is CRC bit part.

4.4.4 Comparative Analysis

The proposed CRC verifier outperforms all other existing faulty signature identification schemes. We have not considered the schemes which require the count of faulty signatures at the beginning of verification since in real time scenario, it is difficult to know the number of faulty signatures beforehand.

The proposed scheme outperforms the existing schemes in terms of computation time to identify the bad signature/s. The number of operations in the proposed scheme to locate the invalid signature is independent of the number of invalid signatures. Therefore we discuss the number of operations the CRC based scheme performs in best, average and worst cases in detail.

Best Case

The best case scenario happens when all the signatures are valid, and there are no invalid ones. In such a scenario, no more verifications are needed to identify any bad signature after GT. Hence the number of CRC verifications is zero.
No. of GT: 1

Average Case

The average case occurs when one or more signatures are faulty. Suppose there is a batch of signatures of batch size t, and n signatures among them are faulty, then the CRC verifier verifies every signature to locate the faulty ones, because it is not known beforehand the value of n.
No. of GT: 1
No. of CRC individual verifications: t

Worst Case

This is a scenario where all the signatures in a given batch are faulty. For a batch of t signatures, all t signatures are faulty, ie., $n = t$. Hence in the CRC based scheme, after

the failure of GT, every signature is verified using CRC verification. Hence the total number of operations remains the same in the average case and worst case.
No. of GT: 1
No. of CRC verifications: t

The number of operations needed to locate the bad signatures in various other techniques is already discussed. Next, we study the verification time taken by these techniques in doing the same.

4.4.5 Results for CRC based scheme

In case of CRC based verification, the standard CRC check-bits are either 16 or 32 bits. Here in the proposed scheme, an extra 32 bits are appended to the signature s. Hence the signer needs an extra time to generate CRC check-bits and the codeword. We have implemented the sequential verifier, DC verifier, Hash based verifier and CRC verifier, to compare the verification time of the schemes.

We have considered ECDSA* signature scheme for our experimentation. Initially, we perform batch signature verification of ECDSA* signatures using the scheme proposed in the Chapter in 3. If the batch verification fails, then we identify the index of the bad signature using hash verification as well as other existing verifiers. We made a comparative study of the execution time taken by these verifiers to identify the faulty signatures. The experimentation is carried out on a Rock cluster CentOS 6.0 system. The processor is Intel® Xeon® E5-2650. Each machine has twenty cores. And each core runs on 2.3 GHz processor.

Table 4.2: Time taken for Generation operation (msec)

Time (msec)	**Batch Size of Signatures**				
	$\mathbf{2^2}$	$\mathbf{2^4}$	$\mathbf{2^6}$	$\mathbf{2^8}$	$\mathbf{2^{12}}$
Signature Generation	10.2	11	12.06	27.8	148
CRC Generation	1.15	1.27	2.81	8.59	137.4

Table 4.2 gives the values of time needed for a signature generation without CRC and time for CRC codeword generation. The CRC codeword generation adds

a minimum delay at the signer side. But this extra time for CRC generation saves more than 50% of extra verification time in identifying the faulty signature through sequential verification. Hence the generation of CRC code incurs minimum extra time and computation when compared to the amount of time and computation it saves at the verification side.

Next is to analyse how efficient CRC verification is in identifying bad signatures as compared to the existing schemes. We have plotted the total verification time needed to identify the faulty signatures. We have experimented the schemes for various batch sizes.

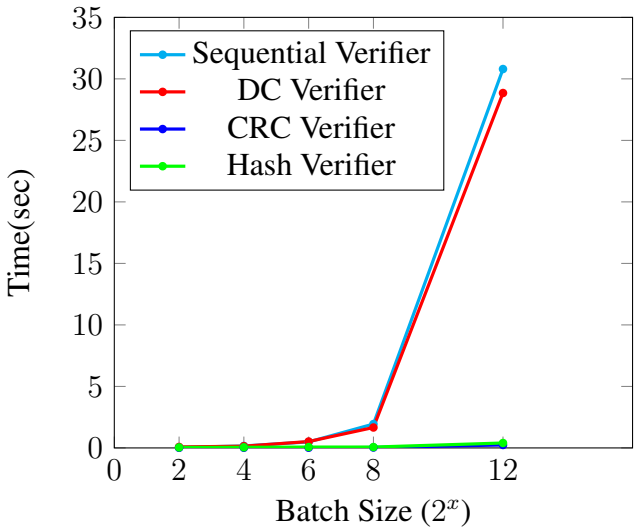

Figure 4.1: Execution Time when 50% signatures are faulty

The two graphs represented in Figures 4.1 and 4.2 indicate the verification time taken by Sequential verifier, DC verifier, Hash verifier and CRC verifier. We can observe clearly from the graphs that, our proposed CRC verifier performs better than the other schemes in both the average as well as worst cases, since it does not involve complex cryptographic computations needed for PKI based signature verification. The proposed CRC based verification scheme performs a little better than the hash based scheme and it involves only binary division operation as the prime operation. Hence the proposed technique proves to be the efficient technique among the available ones.

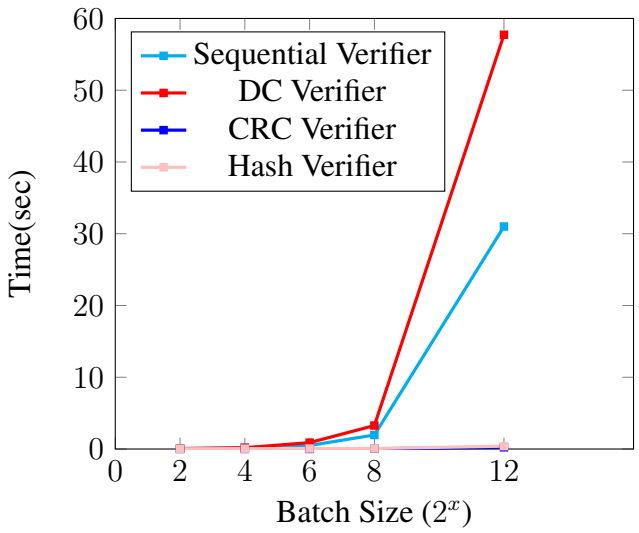

Figure 4.2: Execution Time when all signatures are faulty

4.5 LDPC BASED BAD SIGNATURE IDENTIFICATION SCHEME

There are two sets of codes in coding theory. First is error detection codes, which just detect the error in transmission and second is error correction codes, which detect and correct the error in transmission. As we have already discussed an error-detection code based CRC verification scheme, our next aim is to propose a scheme based on error-correction codes, that uses LDPC codes.

LDPC codes belong to linear error control codes, used for message transfer through a noisy channel in digital communication. Gallager (1962) was the first to introduce LDPC codes in his Ph.D. thesis in 1962 and were redefined by MacKay and Neal (1996). These codes are a way to transmit data through a noisy transmission channel. None of the error control codes provide 100% error-free transmission, but the probability of error can be minimized. LDPC allows data transmission rates very close to the theoretical maximum (Shannon Limit). There are many applications where LDPC code is currently being applied, such as digital satellite television, ultra-high speed wireless local area networks (WLAN), optical communications, and hard disk drives, etc.

In a digital communication system, the purpose of the error correcting codes is to add redundancy to the binary data stream to combat the effect of signal degradation in

the channel. Ideally, channel codes should meet the following requirements:

- Channel codes should be high rate to maximize data throughput.
- Channel codes should have good Bit Error Rate (BER) performance at the desired Signal-to-Noise Ratio(s) (SNR) to minimize the energy needed for transmission.
- Channel codes should have low encoder and decoder complexity.

In our proposed scheme to identify the bad signature, we have used LDPC codes. LDPC codes are generated at the signer, using the Parity-check matrix H. The LDPC code (n, k), where n is the length of the codeword, and k is the message length that needs to be transmitted. LDPC codes can be represented in two ways: Matrix representation and the Graphical representation.

4.5.1 Designing the Parity Check Matrix

The Parity check matrix plays a major role in the performance of LDPC encoding/decoding. Depending on the platform where the encoding/decoding process is done, this matrix can be random or structured. The low-density parity-check matrix H is of order $(n - k) \times n$ for (n, k) code, where n is the length of the codeword and k is the length of signature to be sent. The number of 1's in the parity matrix should be less or sparse. The number of 1's in each row is w_r and the number of 1's in each column is w_c. The conditions $w_r \ll k$ and $w_c \ll n$ must be satisfied for the matrix to be called as low-density. Hence to satisfy this property, the parity check matrix should be sufficiently large. It also requires less memory to store the matrix. There are many ways to generate parity-check matrix.

We are using Gallager (1962) algorithm for generating parity-check matrix. In the Gallager (1962), the parity-check matrix H, for LDPC codes mostly contains 0's and a very few 1's. A regular LDPC code (n, w_c, w_r) of block length n has a parity-check matrix of order $(n - k) \times n$, where the number of 1's in each column is a fixed small number $w_c \geq 3$ and the number of 1's in each column w_r, is also fixed small number $w_r \geq w_c$. Let n be the length of the codeword and k be the length of the signature

to be sent and $m = n - k$ is the length of the parity-check equation. The steps for constructing parity-check matrix are as follows:

- Construct the matrix of order $m \times n$ with w_c number of 1's in each column and w_r in each row.
- Divide the $m \times n$ matrix into $(w_c * m)/(w_r * n)$, with each having a single 1.
- 1's are assigned to the sub-matrices in descending order of sub-matrices such that, the i^{th} sub-matrix contains 1's at positions $(i-1)w_r + 1$ to $i * w_r$.
- Similarly other sub-matrices are designed with similar column permutations.
- Check if no two-columns have an overlap of more than one to avoid length-four cycles.

For our reference, we are considering an example matrix of order 4×8 for (8,4) LDPC code. We can observe that the number of 1s in each row is same. Similarly number of 1s in each column is consistent.

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

The graphical representation was introduced by Tanner (1981) in 1981 for the LDPC codes. LDPC codes are represented using bipartite graphs, where the there are two distinctive sets of nodes which are connected by edges. The edges do not connect the nodes belonging to the same set.

Figure 4.3 represents the bipartite graph for the given code. C-nodes are the check nodes, which represent the parity bits and V-nodes are variable nodes which represent the codewords. In the Figure 4.3, the c_1, c_2, c_3, c_4 are check nodes and $v_1, v_2, \ldots, v_8$ are variable nodes.

An LDPC code can be either regular or irregular based on two conditions. First is if w_c is constant for every column and second is if $w_r = w_c * \frac{m}{n}$. If the matrix satisfies

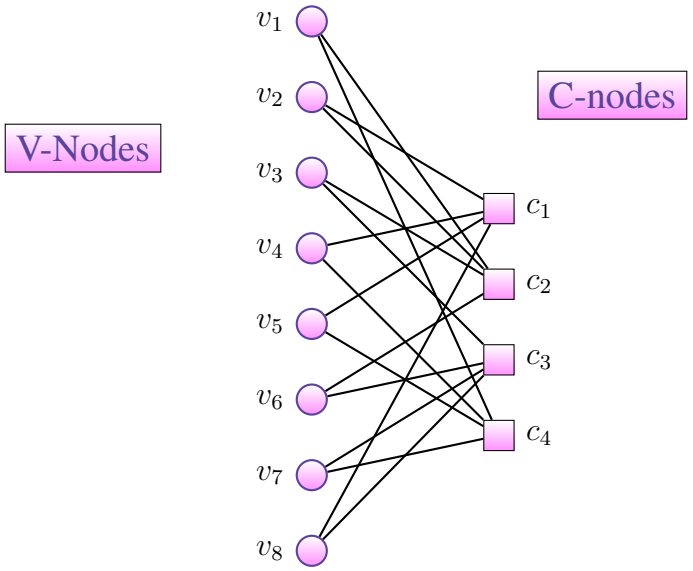

Figure 4.3: Graphical Representation

these conditions, then its a regular one. Hence the example we considered is a regular graph.

There are multiple decoding algorithms available for LDPC codes. The codeword is generated using the generator matrix, which in turn is generated using the parity-check matrix. We are using Gallager's method for generating the parity-check matrix. Similarly, at the receiver side, the received codeword is decoded using Belief Propagation Algorithm (Fossorier et al. 1999).

The decoding part is categorized in to two parts: Hard Decision Decoding and Soft Decision Decoding. In hard decision decoding, the decoder accepts the incoming stream of data as input and considers each bit as definitely either as 1 or 0. The hard decision decoding decides the bit based on the voltage value received, and compares with the threshold, and accordingly makes a decision of 1 or 0. There is more scope of error in hard decision decoding.

The soft decision decoding accepts the stream of data as input and decodes them by taking into account varied combination of values that it can take. Depending on the reliability on the received data, it makes the most favourable decision to form the better estimate of the received input. Since we are using decoding for verification which does not involve any hardware and make decision depending on the computation, soft

decision decoding is the preferable one in our case. Hence we prefer soft-decision decoding in our case.

4.5.2 LDPC Verification Algorithm

There are two algorithms for LDPC based bad signature identification. One is for LDPC based signature generation as shown in Algorithm 4.13 and the other is for LDPC based signature verification. The signer generates the signature as well as LDPC codeword which is verified for every signature to identify the faulty signature at the verifier.

Algorithm 4.13: LDPC based Signature Generation

Input: Secret Key sk and message $m_1, m_2, \ldots, m_t$

Output: $((m_1, s_1), (m_2, s_2), \ldots, (m_t, s_t))$ and LDPC codewords $c_1, c_2, \ldots, c_t$

1. For a given message m to be sent across to the receiver, first the signature s for the message of length k bits.
2. Generate the parity-check matrix H as explained in Section 4.5.1.
3. The parity-check matrix can be calculated by performing Gauss-Jordan elimination on the H to obtain in the form, $H = [A, I_{n-k}]$, where A is binary matrix of order $n - k \times k$ and I_{n-k} is the identity matrix.
4. The generator matrix G can be found as, $G = [I_k, A^T]$, where A^T is the transpose of A.
5. The generator matrix is orthogonal to the parity-check matrix and can be given as, $GH^T = 0$
6. The codeword c is generated for the signature s as, $c = sG^T$, where G^T is the transpose of the Generator matrix and s is the signature.

The number of 1's in rows and columns in parity check matrix H, as defined earlier are represented by w_r and w_c respectively. There is one more important parameter involved in the transmission of a message to the receiver known as Code Rate. Code rate can be defined as the ratio of signature length k to the length of the codeword n.

$$CodeRate = \frac{k}{n}$$

Code rate can also be stated as the amount of actual message transmitted per block. Higher the rate is, greater is the data length to be transmitted. We need the k value to

be higher. Hence the rate should satisfy the condition,

$$\frac{k}{n} > 1 - \frac{w_r}{w_c}$$

Hence to obtain higher code rate, the values of w_r and w_c should be minimum, so that the parity matrix H is as sparse as possible. Hence to decrease the value of n, we increase the value of i in $i * w_c$, where i is $1, 2, \ldots, k$. Suppose to obtain a code rate $\frac{1}{4}$, we chose w_r and w_c as,

$$n = 4 * k - 2 * w_c$$

There are multiple decoding algorithms to decode LDPC codes, independently proposed by various researchers with different names. Some of the popular ones are Belief Propagation Algorithm, Message Passing Algorithm, Sum-Product Algorithm, etc. The most popular decoding scheme is the Belief Propagation (BP) Algorithm. Decoding of LDPC codes is NP-hard for most of the schemes. The decoding algorithm has the parity matrix H, received codeword c, and the number of iterations max_itr as the inputs. The LDPC based signature verification algorithm is shown in Algorithm 4.14.

Algorithm 4.14: LDPC based Signature Verification

Input: Public key pk and $(m, s) = ((m_1, s_1), (m_2, s_2), \ldots, (m_t, s_t))$ and CRC codewords $c_1, c_2, \ldots, c_t$

Output: Index of bad signature

1. Stopping case: If GT$(m, s, 1) = 1$, then accept the signature, else reject.
2. If GT$(m, s, t) = 1$, then accept all the signatures as valid and stop, else go to next step.
3. Perform iterative decoding on received codeword with max_itr as maximum number of iterations using BP algorithm.
4. Assume c' is obtained codeword after decoding.
5. Check obtained c' is a correct codeword or not by computing $H * c'$. If $H * c' = 0$; c, is correct codeword, otherwise remove respective faulty signature/s from the batch and perform batch verification again.

The encoding and decoding steps are explained in Figure 4.4. We can observe that the LDPC encoder generates the codeword for the signature at the signer. Then the

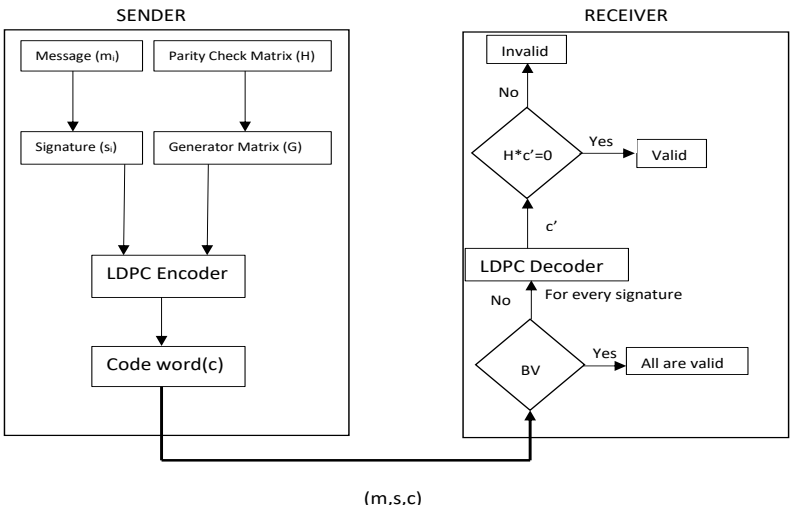

Figure 4.4: LDPC encoding and decoding at sender and receiver

signer sends the message, signature, and the codeword to the verifier. The verifier first performs batch verification, which is shown as BV. If the verification test fails, then there are one or more faulty signatures in the batch. Now the verifier performs individual LDPC verification through LDPC decoder. The decoder generates the codeword c' by iterative BP decoding. This c' is multiplied with parity-check matrix H, if the result leads to zero then the signature has not been modified and is valid, else is invalid. The runtime results for LDPC codes are provided in the next subsection. We have not provided the comparative results for LDPC because, LDPC codes require more time for encoding and decoding as compared to other schemes.

4.5.3 Security Analysis

In case of Man-in-the-Middle attack, if the attacker modifies the signature intentionally or unintentionally, the LDPC verification test returns the signature as faulty. When an attacker modifies one or more bits of signature, the codeword decoding ends up with a positive integer, which results in a conclusion that the signature is faulty.

If an attacker modifies the signature, then the received codeword is c'. Then the batch verification test returns $False$. Then we perform LDPC verification and verify,

$$Hc' \neq 0$$

where H is the parity-check matrix and c' is the received codeword. If the codeword is

modified, then the modular multiplication operation does not lead to remainder zero to reveal the modified signature.

The next attack can be the modification of LDPC bits itself. But the attacker does not know the parity-check matrix, which is shared between the signer and verifier beforehand. Hence without the knowledge of it, it cannot generate the LDPC code bits for the signature. Hence the attacker does not even know the length of LDPC bits and signature. Thus the attacker does not exactly know which part of the codeword is signature and which part is LDPC code bit part.

4.6 RESULTS

In this section, we discuss the results obtained by implementing LDPC codes in software to verify the authenticity of the received message and signature. In the previous section, we have already provided details of CRC codes and hash verification, used for identifying bad signatures. We chose LDPC codes for our experimentation because they are one of the most efficient block coding schemes.

We provide the encoding and decoding times of all three schemes (Hash, CRC, LDPC) separately for different number of signatures. We even provide the signature length and the codeword length for different schemes. We have recorded the encoding and decoding times for more than one signatures.

Table 4.3: CRC Encoding and Decoding times

No. of Signatures	k	n	Encoding (sec)	Decoding (sec)
1	64	80	21×10^{-3}	0.7×10^{-3}
2	128	144	27×10^{-3}	1.3×10^{-3}
3	192	208	36×10^{-3}	2.8×10^{-3}
4	256	272	48×10^{-3}	3.5×10^{-3}

Table 4.3 provides the implementation details for identifying faulty signature using the CRC verification scheme. Similarly we have also provided results of hash encoding and decoding in Table 4.4.

Table 4.5 indicates that the LDPC encoding and decoding times take more time

Table 4.4: Hash Encoding and Decoding times

No. of Signatures	k	n	Encoding (sec)	Decoding (sec)
1	64	224	44×10^{-3}	43.35×10^{-3}
2	128	288	63×10^{-3}	64.35×10^{-3}
3	192	352	75×10^{-3}	75.35×10^{-3}
4	256	416	116×10^{-3}	117.38×10^{-3}

Table 4.5: LDPC Encoding and Decoding times

No. of Signatures	k	n	Encoding (sec)	Decoding (sec)
1	64	248	0.0202	6.3432
2	128	504	0.0241	24.4648
3	192	760	0.0649	67.8917
4	256	1016	0.1312	83.67

than the times for other two schemes. The most time consuming stage in encoding of LDPC codeword is the Gallager's parity-matrix generation. And in the decoding, it is the iterative decoding stage to reconstruct the codeword using the BP algorithm which consumes the maximum time.

The reason for the higher decoding time for LDPC is explained as follows:

- LDPC encoding is matrix multiplication of $v(1, k)$ and $G(k, n)$. The size of one signature is 64 bits, which leads to 64×248 matrix multiplication which is costly.

- Decoding uses Belief propagation algorithm, which uses graph/tree for sum-product message passing. Hence LDPC decoding is computationally heavy operation compared to other implementations.

- Our implementation is at the software level. Hardware-level implementation of encoders and decoders is much faster than software. Parallel / Partially_parallel decoders can be designed for efficient decoding.

4.6.1 Results for the proposed batch verification scheme

In Chapter 3, a new batch verification scheme is introduced which performs verification of ECDSA* signatures in batches. If the scheme returns $True$, then all the signatures are valid, else we apply one of the schemes proposed in this chapter, to identify the faulty signature. In this subsection, we provide results of various batch verification schemes along with the bad signature identification scheme. First, we provide the performance of the hash-based bad signature identification scheme applied after the batch verification test fails.

Table 4.6: Verification Time for Hash-based verification for Single Signers (sec)

Batch Size	Sequential	Naive ECDSA	Naive ECDSA*	S1	S2	Our scheme
2	0.128	0.076	0.076	0.076	0.075	0.077
3	0.528	0.214	0.205	0.211	0.213	0.212
4	0.749	0.254	0.224	0.305	0.254	0.238
5	0.946	0.306	0.234	0.645	0.337	0.290
6	1.223	0.358	0.263	2.396	0.586	0.342
7	1.696	0.411	0.308	6.835	4.135	0.404
8	2.293	0.497	0.363	-	-	0.567
16	9.251	0.962	0.731	-	-	1.023
32	28.71	1.499	1.178	-	-	1.914
64	78.39	2.798	2.335	-	-	3.411

Tables 4.6 and 4.7 refer to the hash-based verification scheme. Table 4.6 is for the single signer, where all the messages are signed by the single signer. We can observe that the proposed batch verification scheme for ECDSA* signatures perform better than S1 and S2 and also than individual verification. Similarly Table 4.7 provides results for all batch verification schemes for ECDSA* signatures, where the signatures in the batch are signed by multiple signers.

Similar to the previous results for Hash-based verification, in the results of CRC based verification to identify faulty signatures also, our batch verification schemes perform better than S1 and S2. There is an important observation in Tables 4.6, 4.7, 4.8 and 4.9 that, the batch verification schemes including our proposed scheme perform better than individual verification even after combining with bad signature identification. Individual verification helps in finding the faulty signature. Similarly,

Table 4.7: Verification Time for Hash based verification for Multiple Signers (sec)

Batch Size	Sequential	Naive ECDSA	Naive ECDSA*	S1	S2	Our scheme
2	0.128	0.100	0.103	0.107	0.100	0.110
3	0.528	0.38	0.369	0.386	0.380	0.419
4	0.749	0.390	0.379	0.585	0.550	0.577
5	0.936	0.658	0.600	1.025	0.680	0.700
6	1.213	0.868	0.755	2.96	1.04	0.893
7	1,686	1.415	1.005	5.48	1.887	1.229
8	2.293	1.830	1.360	-	-	1.654
16	9.291	7.34	5.200	-	-	6.440
32	28.74	21.22	15.41	-	-	19.57
64	79.52	55.43	41.34	-	-	52.92

Table 4.8: Verification Time for CRC based verification for Single Signers (sec)

Batch Size	Sequential	Naive ECDSA	Naive ECDSA*	S1	S2	Our scheme
2	0.134	0.82	0.082	0.082	0.081	0.083
3	0.534	0.220	0.201	0.227	0.216	0.218
4	0.757	0.262	0.232	0.323	0.264	0.246
5	0.951	0.317	0.255	0.666	0.358	0.311
6	1.234	0.369	0.301	2.397	0.617	0.372
7	1.706	0.431	0.338	6.855	4.145	0.434
8	2.326	0.540	0.426	-	-	0.630
16	9.343	1.034	0.803	-	-	1.105
32	28.897	1.686	1.365	-	-	2.101
64	78.84	3.226	2.490	-	-	3.835

batch verification followed by the bad signature identification scheme also identify the faulty signatures. Hence individual verification is proved to be inefficient in most of the cases.

Tables 4.10 and 4.11 represent LPDC based bad signature identification scheme performed after batch verification. The timing results include batch verification time as well as LDPC based bad signature identification time. From all the tables, it is clear that, in IoT application, where we need lightweight yet secure protocols, batch verification along with one of the efficient bad signature identification schemes proves to be efficient than sequential verification. From the results, it is clear that LDPC based verification is not suitable for IoT applications. Instead this verification is suitable for

Table 4.9: Verification Time for CRC based verification for Multiple Signers (sec)

Batch Size	Sequential	Naive ECDSA	Naive ECDSA*	S1	S2	Our scheme
2	0.134	0.106	0.109	0.113	0.106	0.116
3	0.534	0.386	0.375	0.392	0.386	0.425
4	0.757	0.525	0.520	0.593	0.558	0.585
5	0.957	0.679	0.621	1.046	0.701	0.721
6	1.234	0.939	0.776	2.981	1.061	0.914
7	1.706	1.435	1.025	5.5	1.907	1.249
8	2.326	1.863	1.393	-	-	1.707
16	9.343	7.392	5.252	-	-	6.492
32	28.897	21.377	15.567	-	-	19.737
64	78.814	55.724	41.714	-	-	53.294

Table 4.10: Verification Time for LDPC based verification for Single Signers (sec)

Batch Size	Sequential	Naive ECDSA	Naive ECDSA*	S1	S2	Our scheme
2	6.679	7.365	6.627	6.627	6.626	6.628
3	26.022	25.689	25.689	25.715	25.716	25.706
4	68.712	68.217	68.187	68.278	68.219	68.201

applications which have high computation power.

Table 4.11: Verification Time for LDPC based verification for Multiple Signers (sec)

Batch Size	Sequential	Naive ECDSA	Naive ECDSA*	S1	S2	Our scheme
2	6.653	6.625	6.628	6.632	6.625	6.635
3	25.986	25.838	25.827	25.844	25.838	25.877
4	68.664	68.432	68.427	68.500	68.465	68.492

4.7 SUMMARY

This chapter presented one hash based and two error control code based schemes (CRC & LDPC) to identify the location of the bad signature in a given batch of digital signatures during verification. We have provided algorithms for various existing schemes. First we studied hash based verification scheme, which proved to be very efficient in identifying the faulty signatures among the existing schemes. To further the reduce the verification time, we proposed schemes based on error control codes.

CRC based bad signature identification scheme is very efficient compared to LDPC scheme. Because LDPC scheme involves complex operations such as large

matrix multiplication which are time-consuming as well as compute-intensive, another disadvantage with the LDPC code based scheme is that it can not identify the invalid signature until there is major bit difference in the codeword.

Hence in IoT environment, CRC proves to be efficient and secure compared to the existing schemes. Also, the CRC scheme does not need to know the number of bad signatures beforehand. Hence it is practically implementable. Therefore the next aim is to implement the batch verification to identify faulty signatures in an IoT environment. Hence the next objective is to design a trust model which minimizes the bottleneck at the gateway node efficiently without compromise in security.

Chapter 5

A TRUST MODEL BASED BATCH VERIFICATION OF DIGITAL SIGNATURES IN IOT

The security in IoT is one of the trending research areas. Because of the various challenges an IoT network faces as discussed in Chapter 1, designing a standard secure protocol has always attracted the attention of researchers. Hence our aim is to design a lightweight and secure batch verification scheme to verify and identify the bad signatures in a batch of digital signatures. In Chapter 3, we designed a lightweight and secure batch verification scheme which performs better than other schemes and is also secure against most of the attacks. We also developed schemes to identify the bad signatures from a batch that fails the batch verification test in Chapter 4.

The goal of the next research is to reduce the bottleneck at the gateway node by implementing batch verification in IoT network. Hence in this chapter, we have designed a trust model for IoT. The proposed trust model chooses a few trusted sensor nodes carefully based on the trust parameters and shares the verification workload with these nodes. These nodes, in turn, verify the signatures through the batch verification technique and send back the results to the gateway node, which is responsible for further processing.

Trust Model

Trust is the subjective probability level of assessment of one's influence over another's outcome for a particular action or impact on another's performance in a

given situation. There have been many proposals for building computational trust and reputation models (Sabater and Sierra 2005). Trust can be assessed in various ways depending on the application.

Trust models are characterized into:

1. Cognitive: For models based on this approach, trust and reputation are based on the degree of belief. The model also considers the mental state, which leads to the trust on another entity (Esfandiari and Chandrasekharan 2001).

2. Neurological: In such models (Wang et al. 2007), the trust is related to the experiences with external sources from both cognitive and affective perspective.

3. Game-theoretical: The trust and reputation are considered subjective probabilities where an individual A's trust on individual B depends on how B performs for a given action by A (Gambetta et al. 2000).

Our model is Game-theoretical, where the gateway node's trust over the sensor nodes depends on the experience, ie., the experience based on the direct interaction of the gateway node with the sensor nodes. There are various parameters which influence the trust of gateway node on the sensor nodes. In the proposed model, the trust value is calculated based on the previous performances of the node and decided by the gateway node.

The contributions of this chapter are as follows:

1. Proposed a model for the selection of sensor nodes for batch verification of digital signatures based on physical and security parameters of sensor nodes.

2. Introduced a novel reputation-based trust model to select the trusted nodes among the available nodes.

3. Parallel implementation of batch signature verification defined in Section 3.2 using ECDSA* algorithm.

The chapter is organized as follows: Section 5.1 explains the groundwork needed for further understanding. Section 5.2 introduces the proposed model, and Section 5.3 and 5.4 describe the details of the algorithms used in the model. Then Section 5.5 provides experimental results and the chapter is summarized in Section 5.6.

5.1 PRELIMINARIES

In this section, we provide some of the preliminary details to understand the proposed model in a better way. First, we provide the specifications of a few popular sensor nodes followed by the specification of one of the famous gateway nodes.

5.1.1 IoT network nodes

IoT network has sensor nodes and gateway nodes. Multiple sensor nodes send their data at regular intervals to the gateway node. We highlight the specifications of two popular sensor nodes:

1. LOTUS (Frank 2013)
    - **Processor:** Cortex M3 CPU 32-bit processor with 10 - 100MHz
    - **Memory:** 64KB SRAM, 512 KB FLASH, 64MB Serial FLASH
    - **Radio:** Integrated 802.15.4 Radio with on-board 2.4 GHz Antenna
    - **Radio Throughput:** 250 kbps, High Data Rate Radio
    - **Battery:** 2.7-3.3V, 2x AA battery
2. Waspmote (Pham 2014)
    - **Processor:** 8-bit Microcontroller with 14.7456MHz
    - **Memory:** 8KB SRAM, 128 KB FLASH
    - **Radio:** XBee Pro 802.15.4 Radio with 2.4 GHz Antenna
    - **Radio Throughput:** 100 kbps
    - **Battery:** 3.3-4.2V, 6600 mAh Li-Ion rechargeable // 52000 mAh non-rechargeable

The gateway nodes have higher computation power as well as a memory when compared to sensor nodes. Raspberry Pi is one of the famous gateway nodes used in IoT network. Hence we are providing the specifications of Raspberry Pi3 Model B+ (Pi 3)

Raspberry Pi

- **Processor:** ARM Cortex-A53 64-bit quad core processor SoC @ 1.4GHz
- **Memory:** 1GB LPDDR2 SDRAM, 512 KB L2 Cache
- **Connectivity:** 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE Gigabit Ethernet over USB 2.0 (maximum throughput 300Mbps)
- **Power:** 5V/2.5A DC via micro USB connector Power over Ethernet (PoE)–enabled (requires separate PoE HAT)

5.2 PROPOSED MODEL

The gateway nodes as specified in Section 5.1 have higher computational power as compared to sensor nodes, and they also handle many responsibilities such as data aggregation, data preprocessing, authenticity and security of underlying nodes, etc. Therefore it is essential to minimize the load on the gateway nodes. Even though gateway nodes have computation power and energy, but it is not sufficient for complex cryptographic functions. Since the computational capability of the gateway nodes is low, and they have low memory, it is challenging to provide a secure protocol which protects them from most of the attacks. Hence in such an environment, it is better to seek for lightweight cryptographic algorithms for verifying authenticity, encryption and decryption, and other cryptographic operations. Hence we are adopting the batch verification algorithm proposed in Section 3.2, in the IoT environment, where multiple sensor nodes are sending a large number of messages to the gateway node. Thus the gateway node can utilize the advantages of batch verification algorithms to reduce the computation cost and time. Therefore in our proposed model, we have further tried to reduce the computation burden of authentication at the gateway node by distributing the responsibility of authentication of data among the $Trusted$ sensor nodes. The

proposed trust model aids the gateway node in identifying these $Trusted$ nodes and distribute the signatures among the $Trusted$ nodes for verification without compromise in security. These nodes verify the signatures using the batch verification technique already discussed in Chapter 3.

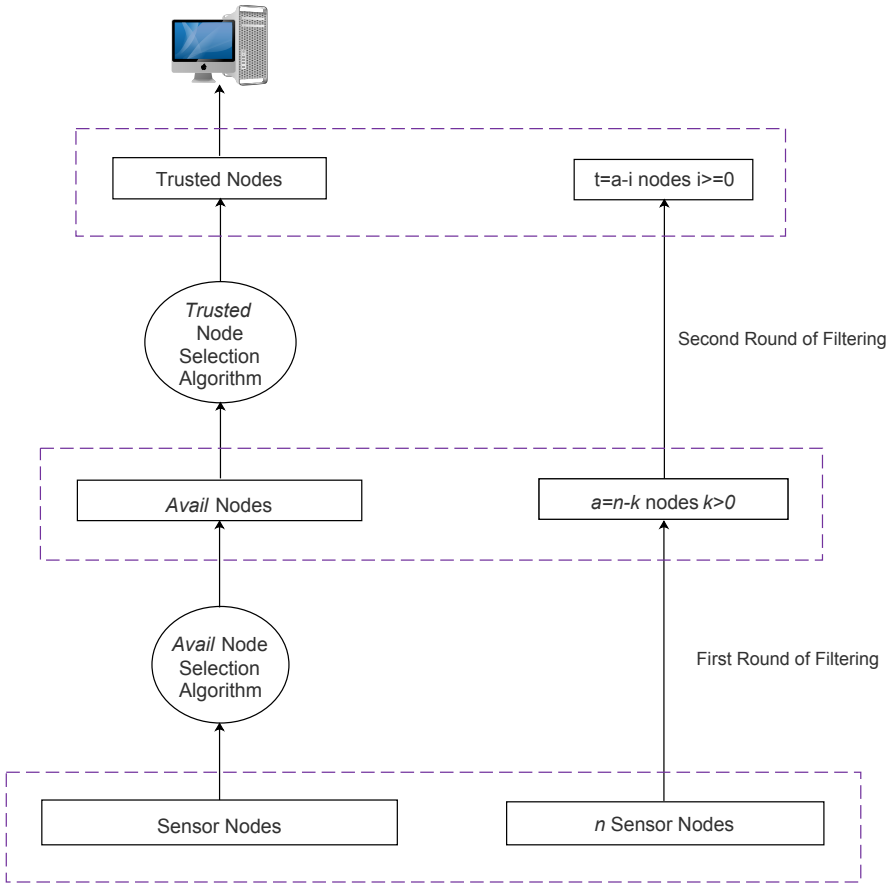

Figure 5.1: Trust Model for Digital Signature Verification in IoT

Figure 5.1 gives an overview of the proposed model. The model aims at reducing the workload at the gateway node by sharing their load with a few trusted sensor nodes. The figure has two algorithms for filtering the sensor nodes and choosing the trusted ones. The first algorithm is for fetching the *Avail* nodes from the set of sensor nodes. The second algorithm is for selecting the set of *Trusted* nodes from the *Avail* node set. Therefore whenever the gateway node in an IoT network receives thousands of signatures which need to be verified, then it uses the first algorithm to identify the available nodes out of all the sensor nodes. Later, to identify the trustworthy nodes from available nodes, the gateway uses the second algorithm.

After choosing the *Trusted* set of nodes at the end of the second algorithm, the next task is sharing of load among these nodes by the gateway node. The gateway node shares a disjoint set of digital signatures with each of these nodes, which perform batch verification on their received set of signatures. Batch verification operation can be performed simultaneously across all the $Trusted$ nodes in parallel. This parallel verification results in a significant reduction in verification time needed to verify multiple signatures. The results for the same are shown later in Table 5.2 and 5.3. These selected *Trusted* nodes perform batch verification on the received batch of signatures. Then the sensor nodes respond the result of the batch verification to the gateway node. The number of *Trusted* nodes chosen depends on the application as well as on the number of signatures to be verified.

There are various parameters considered in our study to choose the trustworthy sensor nodes. These parameters help the gateway node to build trust on the *trusted* sensor nodes. We have designed two algorithms which assist in filtering the nodes. These algorithms are subsequently discussed in Sections 5.4 and 5.5.

5.2.1 Parameters for Node selection

There are multiple parameters of a sensor node which help us to decide whether it is suitable for load distribution. Few of the parameters for node selection are battery capacity level, the efficiency of execution, the type of task the node is already performing, the reputation of the node in history, etc. The threshold value for a few of the parameters is set at the beginning. We have divided the type of parameters into two categories: Physical Parameters and Security Parameters. The two algorithms in our model filter the sensor nodes based on these parameters. The first algorithm considers the physical parameters and generates the *Avail* node set. The second algorithm uses security parameters to generate the *Trusted* node set which is used for load sharing with the gateway node.

Physical Parameters: These are the parameters which define the physical state of the sensor node. In our study, we are considering three physical parameters.

- **Battery Level** helps us decide whether the node can complete the task assigned

to it with the available battery level. The batch verification comes under heavy task for the sensor node, which consumes more power. Hence it is important to check the battery status of the node initially.

- **Type of Task** the node is currently running. If the node is currently running a heavy task like transmission or receiving, then it will end up taking more time to complete the task of batch verification and also will be consuming more energy. Table 5.1 helps us analyze the amount of current discharged during various states of the node. Here we can observe that the node utilizes maximum battery during transmitting and receiving.

Table 5.1: Current Consumption by different states of the node

Node State	***Discharge Current***
Idle	4.8mA
Transmitting	156.2mA
Receiving	138.7mA
Sleep	94.4 μ A

- **Charging status** of the node is one of the essential parameters. If the node's battery level is lower than the threshold and if it is charging, then the probability of the node running out of battery will be less. Hence, in such a scenario, the node can be considered in *Avail* nodes set, provided it satisfies other conditions.

Security Parameters: These parameters help the gateway node in building the trust on the sensor node, hence the name Trust Model. The various security parameters considered are:

- **Availability** checks whether the sensor node is available at a given time.
- **Reliability** indicates whether the node is reliable to perform our task confidentially.
- **Data Integrity** Indicates whether the node is capable of maintaining the integrity of the data being processed.

- **Turnaround Efficiency** indicates how efficiently the node completes the given task in the expected amount of time.

The security parameters are qualitative parameters. Hence we have made an effort to quantify these parameters based on the node's efficiency at completing the given task in history. These parameters are explained in detail in Section 5.5.

As discussed in Liao and Hsiao (2014), for a sensor node with a 5MHz processor, the time per Elliptic Curve Cryptography (ECC) scalar multiplication is approximately 0.06 sec. Hence as per Equation 3.3 and 3.4 in Chapter 3, the number of scalar multiplications needed for the proposed scheme as discussed in Section 3.2, is minimum. Therefore our model does not overload the sensor nodes with batch verification. Hence for sensor nodes in the IoT network, Public Key Infrastructure (PKI) based authentication schemes such as ECC will not create a burden.

5.2.2 Implementation of the ECDSA* batch verification algorithm

Earlier in Chapter 3, we introduced a new batch signature verification algorithm for verifying multiple ECDSA* signatures signed by either single signer or multiple signers. Hence to reduce the verification time at the gateway node, we are using the same batch verification scheme. To further reduce the load on the gateway node, we are performing the batch verification task in parallel among the *Trusted* sensor nodes. We distribute a set of signatures to the *Trusted* nodes which perform batch verification to reduce the verification time. Suppose if there are t signatures received at the gateway node, and there are n *Trusted* sensor nodes, then the received signatures are divided among the sensor nodes. Therefore each node receives approximately $\frac{t}{n}$ signatures.

The sensor node collects all the signatures and verifies them together using the batch verification scheme. If $n > 1$, then all the sensor nodes verify the batch of signatures at the same time. Hence the verification time can be considerably reduced at the verifier. Therefore we performed batch verification in parallel to reduce the verification time compared to the individual verification.

Table 5.2 provides results of running the batch verification code for ECDSA*

Table 5.2: Verification time(sec) for a single signer

Batch Size	Individual Verification	Number of nodes					
		10	20	30	40	50	60
2^2	0.457	0.053	0.048	0.057	0.051	0.055	0.050
2^4	4.737	0.085	0.070	0.068	0.063	0.064	0.072
2^8	6.328	0.707	0.442	0.384	0.309	0.247	0.200
2^{12}	47.914	11.877	5.778	4.35	3.866	3.170	2.559
2^{16}	808.818	190.11	92.334	65.681	57.668	51.399	42.27

Table 5.3: Verification time(sec) for multiple signers

Batch Size	Individual Verification	Number of nodes					
		10	20	30	40	50	60
2^2	0.457	0.064	0.060	0.058	0.061	0.063	0.062
2^4	4.737	0.096	0.089	0.102	0.114	0.101	0.097
2^8	6.328	1.546	0.688	0.552	0.472	0.470	0.460
2^{12}	47.914	25.574	12.470	9.362	7.866	6.82	5.386
2^{16}	808.818	421.88	199.93	150.16	128.26	89.362	65.271

signatures on a different number of nodes for the single signer. We can also observe from the table that batch verification has an advantage over individual verification in case of the higher number of signatures. The verification time can be further reduced by dividing the batch size into smaller sizes, and all are executed in parallel at different *Trusted* sensor nodes, to achieve time efficiency. We can observe that with the increase in the number of nodes working together, the time for batch verification also reduces considerably — similarly, Table 5.3 shows the batch verification results for multiple signers. We can also observe the verification times for 10 nodes and 60 nodes, where the time efficiency is increased by 4.5 times with 60 nodes in case of 2^{16} signatures. Hence more the batch size, the number of nodes should also be more to gain maximum efficiency. Therefore in applications like IoT, after finalizing the *Trusted* nodes, we can make the nodes run in parallel and perform the verification of signatures in batch to gain maximum efficiency.

The batch verification scheme for ECDSA* signatures has been implemented on a cluster system having seven machines, one is master, and the other six are compute machines. The system is a Rock cluster 6.0 system. The processor is Intel® Xeon®

E5-2650. Each machine has ten cores. And each core runs with 2.3 GHz processor. Based on the specification of the sensor node provided in Section 5.1.1, we have simulated each machine as equivalent to 20 sensor nodes. Therefore to simulate the environment of multiple sensor nodes, we used the concept of multi-threading, where a single processor is divided into multiple threads. Hence each thread can be mapped to one sensor node. And we use the MPI library to communicate with other machines in the cluster system for parallel execution. The results show the gain in speedup as we increase the number of nodes. We have considered a maximum of 20 threads per machine.

The gateway node chooses the *Trusted* nodes based on the proposed Trust model for batch verification in IoT network. The results indicate that more the number of nodes, faster are the results. Our next sections brief the algorithms necessary for choosing *Trusted* nodes among the given sensor nodes by the gateway node.

5.3 NODE SELECTION BASED ON PHYSICAL PARAMETERS

The sensor nodes have low computation power and battery. Therefore to share the workload of the gateway node, it is very critical to check parameters of the node such as the node's state and battery level. Therefore these parameters are considered as the physical parameters which define the physical state of the node. We propose an algorithm which chooses the *Avail* nodes depending on the value of these parameters by making a comparison with the threshold value set. Hence it is essential to learn the battery charging and discharging pattern in order to choose the sensor node.

We know that the battery level of the sensor node is an important parameter in *Avail* node selection. Therefore understanding the behavior of the battery charging and discharging is very important. The discharge time can be calculated with the following formula:

$$t = \left(\frac{C}{I}\right) \tag{5.1}$$

where t is the discharge time, C is the battery capacity, and I is the discharge current. We can observe that heavy tasks need more current discharge, as shown in

Table 5.1, which leads to faster battery discharge. Also, the increase in the rate of discharge decreases battery capacity. Therefore the Equation 5.1 holds true in ideal condition. Thus in practical situations, the time of battery discharge Formula (5.2) given by Peukert (1987), considers the decrease in battery capacity aspect also.

$$t = H\left(\frac{C}{IH}\right)^{k} \tag{5.2}$$

where H is the hour rating, k is the Peukert's constant whose value mostly ranges between 1.1-1.3. The Peukert's constant value is usually specified by the vendor on the label of the battery. During the simulation, it is essential to know the battery level of the sensor node before assigning the task of verification by the gateway node. Hence the equation to calculate the available battery capacity at a given time t is given in Equation 5.3,

$$C = e^{\left(\frac{log\frac{t}{H}}{k} + log(IH)\right)} \tag{5.3}$$

The Equations 5.1, 5.2 and 5.3 help us learn the characteristics of battery at any given condition. The gateway node checks the battery level of the sensor node before assigning a task to it. Hence depending on the discharging formula, it can compute whether the sensor node can last with its regular task along with the extra task of verification before the next charging cycle arrives. Therefore the gateway node first computes the battery capacity using Equation 5.3, and then accordingly computes approximate time for which the battery lasts to complete the given task. Hence the threshold battery level needed to assign verification task is set beforehand depending on the kind of application, as well as the capacity of the battery, which aids in filtering the nodes. In the next subsection, we describe our algorithms.

5.3.1 *Avail* node selection algorithm

After checking the battery level of the sensor node, it is important to check whether the node is connected to the power source or no and also to check the type of task the node is running. These parameters can be explained through the Algorithm 5.1 in this subsection. The gateway node checks all the parameters mentioned in Subection 5.3 before making the selection.

The Algorithm 5.1 explains how the *Avail* nodes are selected from the given set of

Algorithm 5.1: *Avail* node selection Algorithm

Input: List of n sensor nodes
n' nodes are not connected to a power source for charging
n'' nodes are connected to a power source for charging
Output: *Avail* nodes

```
1 Case 1: Nodes whose battery is not connected to a power supply are n'
2 for i ← 1 to n' do
      if B_i > B_T then

        if B_status ≠ busy then
          add to Avail list
          B_status = busy
        end if
      end if
3 end
4 end for
5 Case 2: Nodes whose battery is connected to Power supply are n''
6 for j ← 1 to n'' do
      if B_status ≠ busy then
        add to Avail list
        B_status = busy
      end if
7 end
8 end for
```

sensor nodes. We first divide the algorithm into two cases, one is for the nodes which are not charging, and other is for the nodes which are charging.

The first case in the algorithm considers nodes which are not connected to power supply for charging. We initially check the battery level of every node, which is not connected to power supply. If the battery level B_i is below the threshold battery level B_T, i.e., $B_i < B_T$, then we will discard the node. The threshold level is set according to the type of battery as well as the application where it is deployed. In our experimentation, we consider a threshold battery level as 30% of the total battery capacity. If the battery level is more than the threshold, then we check the state of the node, else discard the node. Once the battery capacity of the node is greater than the threshold, then the gateway node checks for the kind of task the node is currently performing. If the node is involved in the task of either transmitting or receiving, i.e., $B_{status} = busy$, then the node is considered busy, and it will not be considered further

in the *Avail* node list. Otherwise, it will be added to the *Avail* list.

In the second case, where the nodes are connected to the power supply, we only verify the status of the node. Since the nodes are connected to the power supply, there is very less probability for the node to run out of battery, since the rate of charging is faster than the rate of discharge. Hence now depending on the kind of the task the node is performing, the decision is made whether to consider the node in *Avail* node list. Thus if the node is not busy in either transmitting or receiving, we include the node in *Avail* list of nodes.

5.4 NODE SELECTION BASED ON SECURITY PARAMETERS

This section aims at choosing *Trusted* nodes by the gateway node from *Avail* nodes for load sharing based on Quality of Service (QoS) value. We use the terms QoS value and trust value interchangeably, but both of them refer to the same value. The designed model aims at evaluating the QoS for every node which can be given as,

$$QoS = w1 * (AV) + w2 * (RL) + w3 * (DI) + w4 * (TE) \tag{5.4}$$

The QoS value computation as shown in Equation 5.4, is determined by Availability (AV), Reliability (RL), Data Integrity (DI) and Turnaround efficiency (TE). The constants $w1, w2, w3$ and $w4$ are fixed at the beginning depending on the kind of application the model is being deployed in and the security requirement. The sum of $w1, w2, w3$, and $w4$ should be equal to one. Suppose in case of cloud computing applications, the choice of $w1 = 0.2, w2 = 0.2, w3 = 0.5$ and $w4 = 0.1$. In our application, the values of the constants are $w1 = 0.25, w2 = 0.4, w3 = 0.1$ and $w4 = 0.25$. The choice of constants can be explained as: since verification of the signature is done to avoid any kind of unreliability, we have assigned the highest preference value to $w2$. Since the IoT environment needs authentic results in real-time processing, Availability and Turnaround Efficiency are assigned next preference and at the end comes Data Integrity. The weights provided remain the same for a given application in the IoT network. Hence the choice of weights is an intelligent and crucial task. The variables (AV, RL, DI, TE) associated with every constant weight, changes

according to the verification performance of the nodes. The weights help us decide the preferences to be given to various security parameters for the given application.

Availability: Availability, in our case, is the degree to which a node is operational or accessible when needed for service. The availability factor of a node is the fraction of the number of job requests accepted to the number of job requests received over a given period. Suppose if a node receives R number of job requests and A number of requests are accepted for processing over a given period of T, then the Availability factor is given as

$$Availability(\mathbf{AV}) = \frac{A}{R}$$

Reliability: Reliability is a measure of trust. It is the degree to which the node performs failure-free operations under given circumstances. It is the measure of the number of jobs successfully completed among the accepted jobs. Suppose among A jobs accepted, and only C are successfully completed over a period T, then Reliability can be given as,

$$Reliability(\mathbf{RL}) = \frac{C}{A}$$

Data Integrity: Data Integrity refers to completeness, accuracy, and consistency of data. Data loss or data modification by any unauthorized node leads to loss of data integrity. Data Integrity aims at preventing accidental and unauthorized changes to information. Suppose out of C successfully completed jobs, only for D jobs data integrity is maintained over a period T, then DI value can be given as,

$$Data\ Integrity(\mathbf{DI}) = \frac{D}{C}$$

Turnaround Efficiency: Turnaround time is the time gap between the submission of job and successful completion of the job by the node. The expected turnaround time is the turnaround time specified for the node, and actual turnaround time is the total time between the submission and successful completion of the job in a practical situation. Actual turnaround time is usually different from the expected turnaround time. TE for a given period T can be given as,

$$Turnaround\ Efficiency\ for\ a\ node(\textbf{TE}) = \frac{Expected\ turnaround\ time}{Actual\ turnaround\ time}$$

5.4.1 *Trusted* Node Selection Algorithm

Our trust model concentrates on choosing the *Trusted* nodes among the *Avail* nodes through their QoS value. Every *Trusted* sensor node and gateway node update the QoS value for the *Trusted* node after efficient completion of verification job request. Therefore higher the QoS value, higher the probability of the node getting chosen for load sharing. Thus depending on the need of the number of sensor nodes for load sharing in an application, we decide the size of the *Trusted* node set.

Suppose if there are 500 available nodes, and the signatures to be verified is 2^{15}, then according to our analysis, 100 nodes will provide sufficient speedup. Hence instead of choosing all 500 available nodes, the gateway node chooses only 100 nodes from the *Avail* nodes to distribute the load. Therefore 100 nodes are selected based on their physical and security parameter values. The top 100 *Avail* nodes sorted according to QoS value are chosen. The choice of the number of *Trusted* nodes depends on the number of signatures received at the gateway node. For our experimentation, we have chosen the number of *Trusted* nodes in such a way that each trusted node gets maximum 500 signatures. Hence with these conditions, we can achieve maximum speedup.

Algorithm 5.2 explains the criteria for choosing the nodes for the parallel batch verification by the gateway node. The reason for sorting the nodes according to QoS value is to choose the most trusted nodes. The nodes with higher QoS value indicates that the nodes have a higher success rate for completion of verification. The unsuccessful completion of the task by the node might be due to many reasons. Either the node went down because of some hardware or software crash or natural calamities, or it may be due to sudden node compromise. Hence verifying the behavior and performance of the node in history is very important. The reputation of the node helps us in minimizing the vulnerable node selection probability.

The gateway node distributes a set of different signatures to each of the $Trusted$ nodes. These $Trusted$ nodes perform batch verification to check whether the received

Algorithm 5.2: Scheduling Available gateway Nodes

Input: List of *Avail* nodes **g**
Output: *Trusted* nodes **c**

```
Sort the Avail nodes based on QoS value
Verify the QoS value stored with the node with the value stored in gateway node
for j ← 1 to Avail do
    if(stored[j].QoS≠j.QoS)
        Discard the node from Avail list
end
end for
Choose the top c nodes needed for execution
Send();  Distribute various batch of signatures to c nodes
Receive();  Receive the verification result from individual nodes
for i ← 1 to c do
    Increment AV for node i
    if ( i.status=successfully completed in given turnaround time )
        Increment RL, DI, TE for node i
    else If (i.status=successfully completed with more than expected turnaround time)
        Increment RL, DI for node i
        Decrement TE for node i
    else If (i.status=successfully completed with loss of data integrity)
        Increment RL for node i
        Decrement DI, TE for node i
    else
        Decrement RL, DI, TE for node i
end
end for
```

batch or set of signatures are valid or no. If the batch of signatures fails the verification test, then either the sensor node or the gateway node will verify further to identify the faulty signature/s. If the sensor node notifies that the batch contains bad signatures, then the gateway node identifies the bad signature through any of the schemes, proposed in Chapter 4, to identify the faulty signature/s.

5.5 RESULTS AND DISCUSSION

In this section, we provide the results of implementing our model. We also compare our results in various scenarios with other models. We provide the probability of choosing nodes that can complete the job successfully in our proposed model and other models.

Our first set of results are taken by considering ideal conditions, 1) the battery is 100% efficient, 2) the node never goes down unless the battery level is zero, 3) the nodes with QoS value greater than 30% will never fail. We are considering four models for load distribution:

1. In the **Proposed model**, the nodes are filtered to perform batch verification based on the physical and security parameters. It is explained in detail in Section 5.3 and 5.4.

2. In the **Random Selection model**, the nodes are chosen randomly for performing batch verification. There is no filter on the selection of nodes. Hence this model has a higher probability of node failure.

3. In the **Physical Parameter model**, the nodes are filtered based on the physical parameters and not security. Hence the probability of node failure is reduced as compared to the Random Selection model.

4. In the **Security Parameter model**, nodes are filtered based on only the security parameters. The probability of node failure in this model is comparable to the Physical Parameter model.

We have implemented all the four models for the ideal condition as well as for practical conditions. We will go through the results in both the conditions in separate subsections.

5.5.1 Ideal Condition Results

As we know, Ideal condition is the state where the entire model is 100% efficient. In such a state node battery is fully efficient, and its efficiency does not decrease even if the battery level goes less than the threshold level. To make a detailed study of the four models, we consider an IoT network scenario and compare the efficiency of all the models.

Consider a scenario of an IoT network consisting of 1000 sensor nodes connected to a single gateway node. Therefore to distribute the load from gateway node among

the sensor nodes, the proposed model and other models pick 100 sensor nodes from the 1000 nodes and distribute a set of signatures to each of the sensor nodes. Among the chosen 100 sensor nodes, there is always a probability of node getting dysfunctional because of unintended hardware or software failure. Hence among the 100 nodes chosen, we consider 1-10% of nodes failing randomly due to hardware or software failure. Now the efficiency of the models lie in reducing the intended node failure such as node running out of battery, node getting compromised etc., which can be avoided.

We have performed the experiment 50 times to choose the 100 nodes effectively using every model to compare the efficiency of models in reducing the intended failures. Table 5.4(a) provides the results of 50 iterations for the proposed model. Each number in the table indicates the number of sensor nodes available to the gateway node ($Trusted$) after intended and unintended failures. Therefore the average of these 50 iterations is almost 95. The proposed scheme considerably reduces the intended node failures. Hence because of low node failure, more number of sensor nodes are available for gateway node to share the load and more efficient will be the network.

Similarly we have performed 50 experiments for all the other three models. In the Random selection model, since the 100 nodes are chosen randomly, they will have maximum node failure because of the intended node failures. Therefore from the Table 5.4(b), it is clear that average number of available nodes after intended and unintended node failure is 87.

Table 5.4: Ideal Condition (a) Proposed Model Node Selection (b) Random Node Selection

(a) Available nodes									
93	96	95	98	91	95	97	96	99	99
99	91	99	94	97	92	100	97	92	93
96	91	95	92	95	96	92	93	93	93
96	93	98	91	97	96	93	97	94	100
96	91	98	96	96	96	95	93	100	93

(b) Available nodes									
82	80	79	84	83	84	92	90	87	96
95	94	96	93	93	98	99	93	94	90
82	81	84	82	89	82	87	86	89	88
90	86	90	86	92	92	96	90	94	90
86	82	85	81	89	89	87	89	88	91

Tables 5.5(a) indicates the results for the Physical parameter model, where the 100 nodes are selected based on the physical parameters of the node. This considerably reduces the intended node failure due to lack of battery, or node unavailability due to other scheduled task. Intended failure may be because of node compromise, node

Table 5.5: Ideal Condition (a) Physical Parameter based Node Selection (b) Security Parameter based Node Selection

(c) Available Nodes									
87	83	82	88	84	84	90	97	90	87
95	95	87	86	93	89	88	84	96	95
81	92	91	90	84	89	90	83	95	88
94	87	95	93	93	90	87	94	94	91
84	87	92	90	86	95	93	97	94	92

(d) Available Nodes									
92	91	95	89	94	92	93	93	97	92
94	91	95	87	91	97	89	89	91	93
88	89	89	93	94	89	95	94	94	87
92	88	92	96	94	91	95	94	94	97
90	88	91	88	88	95	92	88	87	89

unresponsive. Therefore the average number of nodes available after intended and unintended node failure is higher than Random selection model but less than the proposed model. The average of the entries in Table 5.5(a) is around 90. Hence an average of 10% nodes fail due to intended and unintended node failure.

On similar grounds, Table 5.5(b) represents the experimental results for 50 iterations of Security parameter model. Each value in the table indicates the number of nodes available for load sharing in each iteration of the model after the node failures. The intended node failure is due to node running out of battery or node being busy with other tasks. Hence the selection of compromised nodes is prevented which reduces the node failure compared to Random selection model. The performance of this model is equivalent to Physical selection model. The average of the 50 iterations is around 90.5.

Hence from the results of all the four models, it is clear that the proposed model reduces the intended node failure probability considerably. The unintended node failures are not in control. Hence the gateway node gets more number of $Trusted$ nodes to share load to reduce the bottleneck at the gateway node and increase the efficiency of the network.

5.5.2 Practical Condition Results

We considered ideal conditions for node selection in the previous subsection, and now we consider the practical conditions. In practical conditions, the battery efficiency reduces when the battery level goes down below 30%(we assumed that the efficiency decreases as the battery level drops below 30% and hence the node can not perform batch verification). Hence we have set the threshold battery level to 30% in Algorithm 5.1. And also the nodes with trust value less than 50%, fall under greater node failure

category. Our proposed model checks for these conditions by default. Hence the performance of our model does not reduce in practical conditions too, whereas it affects the other models.

As explained in the ideal conditions, in practical conditions too, we analyse the results of all the four models, each performed for 50 iterations. For the proposed model, the Table 5.6(a) provides 50 entries which indicate the number of nodes available after node failure at the end of each iteration. There is no change in the performance of the proposed model when compared to ideal condition, since the model avoids the intended node failures.

The results for Random selection model as shown in Table 5.6(b) indicates that the node failure has increased in practical conditions. Since the nodes are randomly picked for load sharing by the gateway node, the intended node failures increase. Hence the model performs worst with an average number of nodes being available equal to 71 by considering results of all 50 iterations. The selected 100 nodes have more number of nodes with battery less than 30% and QoS less than 50%.

Table 5.6: Practical Condition (a) Proposed Model Node Selection (b) Random Node Selection

(a) Available nodes									
93	96	95	98	91	95	97	96	99	99
99	91	99	94	97	92	100	97	92	93
96	91	95	92	95	96	92	93	93	93
96	93	98	91	97	96	93	97	94	100
96	91	98	96	96	96	95	93	100	93

(b) Available nodes									
62	60	59	64	63	64	72	70	67	76
75	74	76	73	63	68	69	73	74	70
62	71	64	72	69	62	67	66	69	68
70	66	70	66	72	72	76	70	74	70
66	62	65	61	69	69	67	69	68	71

Table 5.7: Practical Condition (a) Physical Parameter based Node Selection (b) Security Parameter based Node Selection

Available Nodes									
77	73	72	78	74	74	80	87	80	77
95	95	87	86	93	89	88	84	96	95
71	82	81	80	74	79	80	73	85	78
84	77	85	83	83	80	77	84	84	81
74	77	82	80	76	85	83	87	84	82

Available Nodes									
82	81	85	79	84	82	83	83	87	82
84	81	85	77	81	87	79	79	81	83
78	79	79	83	84	79	85	84	84	77
82	78	82	86	84	81	85	84	84	87
80	78	81	78	78	85	82	78	77	79

The average number of failure-free nodes for Physical parameter model in practical conditions is 80.18. This number is bigger than the Random selection model,

but smaller than the proposed model. This model includes those nodes that are compromised and unresponsive with QoS value less than 50%. Hence the total number of failure-free node's average is smaller than the average of the proposed model. This is shown in Table 5.7(a).

Similar to Physical parameter model, in case of Security parameter model as shown in Table 5.7(b), the average of the entries in the table is 82. The node failure in practical conditions for this model is due to the node having battery less than 30% . The performance of the model in practical conditions decreases because of the restrictions in practical scenario. Hence the node failure probability due to its battery level less than 30% is more.

Hence by comparing the performance of all the four models in practical conditions, we can observe that the proposed model performs better in practical conditions too. The efficiency of the Random selection model decreases significantly because of the restriction in the practical scenario. The proposed model performs better in both the conditions because the model keeps a check on every parameter of the sensor node. Also, the performance does not degrade by varying conditions and by varying the number of sensor nodes. Hence the proposed model plays a crucial role in implementing batch verification in IoT network efficiently. Hence the proposed model also reduces the bottleneck at the gateway without compromise in security.

5.5.3 Security Analysis

We have used ECDSA* signatures (Antipa et al. 2005) for node authentication. ECDSA* signature is as secure as ECDSA signature. And for batch verification, we are using the batch verification scheme proposed in Chapter 3. The security for the trust model is measured through the trust value of the sensor node. Hence in the Algorithm 5.2, we can observe that the gateway node prefers the nodes with higher QoS value.

Suppose in one of the scenarios, a node gets compromised, and its QoS value is illegally incremented to get access to signatures, then the proposed model is efficient to detect it. In such an attack, the attacker tries to include the compromised node in the list of trusted nodes. This way, the attacker tries to verify an illegal or bad signature as

the legal signature by paving way for the other attacks.

The unauthorized manipulation of the QoS value of the node can be detected by storing a copy of QoS value of every node in the gateway node. Hence, once the *Avail* nodes are derived from Algorithm 5.1, our next task is to verify whether the QoS values of these nodes are same as the QoS values stored in the gateway node. Therefore this way, it will be easy for the gateway node to identify the faulty node and discard it from the list. This way, the node will not be considered for further verification task by the gateway node.

For the Data information spoofing during communication between the sensor node and the gateway node, there are multiple Key Predistribution Schemes available (Chan et al. 2003; Du et al. 2005). Since sensor nodes have low bandwidth and computation power, it is imperative to have lightweight protocols like Simple Object Access Protocol (SOAP) (Mitra et al. 2003), Constrained Application Protocol (CoAP) (Shelby et al. 2014), etc. There are other protocols for secure communication such as IPsec (Frankel and Krishnan 2011), DTLS (Rescorla and Modadugu 2012) etc. These protocols are not suitable for communication in sensor nodes since the protocols have a high bandwidth delay product, high packet loss.

There is always a trade-off between the security and the computation time. Many industries come up with technologies which compromise the security in a contest to reduce the computation time. Hence in our trust model, we are trying to reduce the computation load at the gateway node by distributing load across *Trusted* nodes, which reduce the probability of node failure. Our scheme does not increase the latency since the sensor nodes need not have to communicate back the entire batch of signatures to the gateway node. The sensor nodes verify the signatures through batch verification and respond to the gateway node whether the batch verification is a $success$ or a $failure$.

In case of failure of batch verification test, the gateway node can find the exact location of the forged signature by performing the individual verification of signatures by using bad signature identification schemes. This load sharing reduces the computation cost to find the exact index of the forged signature.

5.6 SUMMARY

In this chapter, we have proposed a trust model, which helps in the efficient implementation of batch verification in IoT applications to reduce the load on the gateway node. Since the gateway node has many responsibilities, the proposed model significantly reduces a load of authentication for every message received from hundreds of sensor nodes.

The proposed trust model has two algorithms: one for choosing *Avail* sensor nodes based on physical parameters and the second for choosing *Trusted* sensor nodes from the *Avail* nodes. The gateway node distributes the signatures among the *Trusted* nodes. The *Trusted* nodes perform batch verification and return the result of verification to the gateway node. This significantly reduces the signature verification time and load at the gateway node, which the gateway node can utilize for performing other tasks. Hence the proposed model successfully implements the batch verification in IoT network efficiently.

Chapter 6

CONCLUSIONS AND FUTURE WORK

IoT network consists of sensor nodes, actuators, gateway nodes, etc. These nodes have low computation power, memory, and energy. Among these nodes, the gateway node has more computation power and energy compared to other nodes. All the communication with the external world happens through gateway node, making the gateway a bridge between the IoT network and external cloud. This role creates a bottleneck at the gateway node since the gateway becomes responsible for authentication, verification, normalization of data received from the sensor nodes. Therefore to reduce this bottleneck, there is a need for lightweight, secure schemes which can be implemented in the IoT network to reduce the load on the gateway node.

To develop a lightweight batch verification scheme, it is essential to survey various batch verification schemes available in the literature. Since ECDSA* signature is the lightweight signature algorithm, it is advantageous to develop a batch verification scheme for ECDSA* signatures. Therefore in this work, a new batch verification scheme for ECDSA* signatures is developed, which is secure as well lightweight compared to existing schemes. The proposed batch verification scheme and most of the existing schemes do not identify the bad signature in the batch when the batch verification test fails. Therefore there are various bad signature identification schemes available in the literature to locate the bad signature in the batch. But these available schemes suffer from various drawbacks.

Therefore the second work of the thesis concentrates on developing bad signature

schemes based on the hash function and Error Control Codes. The schemes are applied with batch verification. The signer generates the signature and the codeword and sends to the verifier. If the batch verification test fails, the verifier performs decoding of the codeword to identify the bad signature. We have presented the encoding and decoding times for the three schemes and compare results with other existing schemes. We found that the Hash and CRC based verification are efficient for IoT applications than LDPC based verification.

The next contribution of the thesis is to design a trust model to implement the proposed schemes efficiently to reduce the bottleneck at the gateway node. The trust model identifies the $Trusted$ sensor nodes out of all the sensor nodes. These $Trusted$ sensor nodes verify the received signatures using proposed schemes and return the result to the gateway node. The model introduces two algorithms for choosing the $Trusted$ nodes among the available nodes based on various parameters associated with the nodes.

Future Scope

As the approaches proposed in the thesis reduce the load on the gateway nodes, there is significant scope for future research. Further research can be carried out in the following areas:

- The existing batch verification schemes are not efficient for multiple signers. Hence to gain further efficiency, designing schemes which are more suitable for multiple signers is an excellent research topic to study.

- Most of the existing batch verification schemes verify the received batch of signatures to check if there is a presence of any faulty signature/s. Hence a promising future research direction is to design batch verification scheme which can verify as well as identify the bad signature in the received batch of digital signatures.

- Since ECDSA is grabbing attention in the recent times due to its short signature property, it has the overhead of precomputation. ECDSA* is one of the ways to minimize the precomputation overhead, but it needs extra bits of information,

which increases the signature size, to gain efficiency. Although the extra bits of information in ECDSA* does not create significant overhead, developing techniques as efficient as or more efficient than ECDSA* requires attention.

- The trust model designed reduces the overhead at the gateway node in IoT by distributing the authentication load among the sensor nodes. Hence exploring new schemes to reduce other responsibilities of gateway node and models to reduce heavy computation load is a promising research direction in this area.

In conclusion, this dissertation proposes a new batch verification scheme for verifying multiple ECDSA* signatures, which is more efficient and secure than the existing schemes. One new hash-based and two new error control code based schemes are proposed to identify the faulty signatures in the batch, which are efficient in identifying faulty signature than individual verification. To reduce the overhead of gateway node, a trust model for IoT network is designed. The overhead is reduced by distributing the authentication load to sensor nodes, which perform batch verification to verify the digital signatures.

BIBLIOGRAPHY

Abe, M. (1998). "Universally verifiable mix-net with verification work independent of the number of mix-servers." In *International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 437–447.

Adleman, L., Rivest, R. L. and Shamir, A. (1977). "On digital signatures and public key cryptosystems.").

Amendola, S., Lodato, R., Manzari, S., Occhiuzzi, C. and Marrocco, G. (2014). "Rfid technology for iot-based personal healthcare in smart spaces." *IEEE Internet of Things Journal*, 1(2), 144–152.

Antipa, A., Brown, D., Gallant, R., Lambert, R., Struik, R. and Vanstone, S. (2005). "Accelerated verification of ecdsa signatures." In *International Workshop on Selected Areas in Cryptography*, Springer, 307–318.

Atzori, L., Iera, A. and Morabito, G. (2010). "The internet of things: A survey." *Computer networks*, 54(15), 2787–2805.

Bao, F., Lee, C.-C. and Hwang, M.-S. (2006). "Cryptanalysis and improvement on batch verifying multiple rsa digital signatures." *Applied Mathematics and Computation*, 172(2), 1195–1200.

Bellare, M., Garay, J. A. and Rabin, T. (1998). "Fast batch verification for modular exponentiation and digital signatures." In *International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 236–250.

Bellare, M. and Rogaway, P. (1996). "The exact security of digital signatures-how to

sign with rsa and rabin." In *International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 399–416.

Bernstein, D. J., Doumen, J., Lange, T. and Oosterwijk, J.-J. (2012). "Faster batch forgery identification." In *International Conference on Cryptology in India*, Springer, 454–473.

Blahut, R. E. (2014). *Cryptography and Secure Communication*, Cambridge University Press.

Blake, I. F., Seroussi, G. and Smart, N. (1999). *Elliptic curves in cryptography*, volume 265, Cambridge university press.

Boneh, D. and Boyen, X. (2004). "Short signatures without random oracles." In *Eurocrypt*, volume 3027, Springer, 56–73.

Boneh, D., Boyen, X. and Shacham, H. (2004). "Short group signatures." In *Crypto*, volume 3152, Springer, 41–55.

Boneh, D. et al. (1999). "Twenty years of attacks on the rsa cryptosystem." *Notices of the AMS*, 46(2), 203–213.

Boneh, D., Lynn, B. and Shacham, H. (2001). "Short signatures from the weil pairing." In *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, 514–532.

Brown, D. R. (2005). "Generic groups, collision resistance, and ecdsa." *Designs, Codes and Cryptography*, 35(1), 119–152.

Buzzanca, M., Carchiolo, V., Longheu, A., Malgeri, M. and Mangioni, G. (2017). "Direct trust assignment using social reputation and aging." *Journal of Ambient Intelligence and Humanized Computing*, 8(2), 167–175.

Camenisch, J., Hohenberger, S. and Pedersen, M. O. (2007). "Batch verification of short signatures." In *Eurocrypt*, volume 4515, Springer, 246–263.

Camenisch, J. and Lysyanskaya, A. (2004). "Signature schemes and anonymous credentials from bilinear maps." In *Annual International Cryptology Conference*, Springer, 56–72.

Chan, H., Perrig, A. and Song, D. (2003). "Random key predistribution schemes for sensor networks." In *IEEE symposium on security and privacy*, volume 197, Berkeley, California.

Changchien, S. W., Hwang, M.-S. and Hwang, K.-F. (2002). "A batch verifying and detecting multiple rsa digital signatures." *International Journal of Computational and Numerical Analysis and Applications*, 2(3), 303–307.

Chen, H., Wu, H., Zhou, X. and Gao, C. (2007). "Agent-based trust model in wireless sensor networks." In *null*, IEEE, 119–124.

Chen, T., Wang, J. and Zhou, Y. (2001). "Combined digital signature and digital watermark scheme for image authentication." In *Info-tech and Info-net, 2001. Proceedings. ICII 2001-Beijing. 2001 International Conferences on*, volume 5, IEEE, 78–82.

Cheon, J. H. (2002). "A universal forgery of hess's second id-based signature against the known-message attack.." *IACR Cryptology ePrint Archive*, 2002, 28.

Cheon, J. H. and Yi, J. H. (2007). "Fast batch verification of multiple signatures." In *International Workshop on Public Key Cryptography*, Springer, 442–457.

Claessens, J., Dem, V., De Cock, D., Preneel, B. and Vandewalle, J. (2002). "On the security of today's online electronic banking systems." *Computers & Security*, 21(3), 253–265.

Cocchia, A. (2014). "Smart and digital city: A systematic literature review." In *Smart city*, Springer, 13–43.

Davies, D. W. (1983). "Applying the rsa digital signature to electronic mail.." *IEEE Computer*, 16(2), 55–62.

Delerablée, C. and Pointcheval, D. (2006). "Dynamic fully anonymous short group signatures." *Vietcrypt*, 4341, 193–210.

Diffie, W. and Hellman, M. (1976). "New directions in cryptography." *IEEE transactions on Information Theory*, 22(6), 644–654.

Du, K.-k., Wang, Z.-l. and Mi, H. (2013). "Human machine interactive system on smart home of iot." *The Journal of China Universities of Posts and Telecommunications*, 20, 96–99.

Du, W., Deng, J., Han, Y. S., Varshney, P. K., Katz, J. and Khalili, A. (2005). "A pairwise key predistribution scheme for wireless sensor networks." *ACM Transactions on Information and System Security (TISSEC)*, 8(2), 228–258.

Escolar, S., Chessa, S. and Carretero, J. (2014). "Energy management in solar cells powered wireless sensor networks for quality of service optimization." *Personal and ubiquitous computing*, 18(2), 449–464.

Esfandiari, B. and Chandrasekharan, S. (2001). "On how agents make friends: Mechanisms for trust acquisition." In *Proceedings of the fourth workshop on deception, fraud and trust in agent societies*, volume 222, 19.

Ferrara, A. L., Green, M., Hohenberger, S. and Pedersen, M. Ø. (2009). "Practical short signature batch verification.." In *CT-RSA*, volume 5473, Springer, 309–324.

Fiat, A. (1989). "Batch rsa." In *Conference on the Theory and Application of Cryptology*, Springer, 175–185.

Fisher, R., Ledwaba, L., Hancke, G. and Kruger, C. (2015). "Open hardware: A role to play in wireless sensor networks?." *Sensors*, 15(3), 6818–6844.

Forney, G. (1965). "On decoding bch codes." *IEEE Transactions on information theory*, 11(4), 549–557.

Fossorier, M. P., Mihaljevic, M. and Imai, H. (1999). "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation." *IEEE Transactions on communications*, 47(5), 673–680.

Frank, R. (2013). *Understanding smart sensors*, Artech House.

Frankel, S. and Krishnan, S. (2011). "Ip security (ipsec) and internet key exchange (ike) document roadmap." Technical report.

Furnell, S. M. and Karweni, T. (1999). "Security implications of electronic commerce: a survey of consumers and businesses." *Internet research*, 9(5), 372–382.

Gallager, R. (1962). "Low-density parity-check codes." *IRE Transactions on information theory*, 8(1), 21–28.

Gambetta, D. et al. (2000). "Can we trust trust." *Trust: Making and breaking cooperative relations*, 13, 213–237.

Ganeriwal, S., Balzano, L. K. and Srivastava, M. B. (2008). "Reputation-based framework for high integrity sensor networks." *ACM Transactions on Sensor Networks (TOSN)*, 4(3), 15.

Goh, E.-J. and Jarecki, S. (2003). "A signature scheme as secure as the diffie-hellman problem." In *International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 401–415.

Goldwasser, S., Micali, S. and Rivest, R. L. (1988). "A digital signature scheme secure against adaptive chosen-message attacks." *SIAM Journal on Computing*, 17(2), 281–308.

Gubbi, J., Buyya, R., Marusic, S. and Palaniswami, M. (2013). "Internet of things (iot): A vision, architectural elements, and future directions." *Future generation computer systems*, 29(7), 1645–1660.

Guo, F., Mu, Y. and Chen, Z. (2008). "Efficient batch verification of short signatures for a single-signer setting without random oracles." *Advances in Information and Computer Security*, 49–63.

Harn, L. (1998a). "Batch verifying multiple dsa-type digital signatures." *Electronics Letters*, 34(9), 870–871.

Harn, L. (1998b). "Batch verifying multiple rsa digital signatures." *Electronics Letters*, 34(12), 1219–1220.

Harn, L. and Xu, Y. (1994). "Design of generalized elgamal type digital signature schemes based on discrete logarithm." *Electronics letters*, 30(24), 2025–2026.

Housley, R., Polk, W., Ford, W. and Solo, D. (2002). "Internet x. 509 public key infrastructure certificate and certificate revocation list (crl) profile." Technical report.

Hwang, M.-S., Lee, C.-C. and Lu, E. J.-L. (2001). "Cryptanalysis of the batch verifying multiple dsa-type digital signatures." *Pakistan Journal of Applied Sciences*, 1(3), 287–288.

Hwang, M.-S., Lin, I.-C. and Hwang, K.-F. (2000). "Cryptanalysis of the batch verifying multiple rsa digital signatures." *Informatica*, 11(1), 15–18.

Jie, Y., Pei, J. Y., Jun, L., Yun, G. and Wei, X. (2013). "Smart home system based on iot technologies." In *Computational and Information Sciences (ICCIS), 2013 Fifth International Conference on*, IEEE, 1789–1791.

Jin, J., Gubbi, J., Marusic, S. and Palaniswami, M. (2014). "An information framework for creating a smart city through internet of things." *IEEE Internet of Things Journal*, 1(2), 112–121.

Kalra, S. and Sood, S. K. (2015). "Secure authentication scheme for iot and cloud servers." *Pervasive and Mobile Computing*, 24, 210–223.

Kamvar, S. D., Schlosser, M. T. and Garcia-Molina, H. (2003). "The eigentrust algorithm for reputation management in p2p networks." In *Proceedings of the 12th international conference on World Wide Web*, ACM, 640–651.

Karakostas, B. (2013). "A dns architecture for the internet of things: A case study in transport logistics." *Procedia Computer Science*, 19, 594–601.

Karati, S., Das, A. and Chowdhury, D. R. (2012a). "Using randomizers for batch verification of ecdsa signatures.." *IACR Cryptology ePrint Archive*, 2012, 582.

Karati, S., Das, A., Roychowdhury, D., Bellur, B., Bhattacharya, D. and Iyer, A. (2012b). "Batch verification of ecdsa signatures." In *International Conference on Cryptology in Africa*, Springer, 1–18.

Katz, J. (2010). *Digital signatures*, Springer Science & Business Media.

Katz, J. and Lindell, Y. (2014). *Introduction to modern cryptography*, CRC press.

Katz, J., Menezes, A. J., Van Oorschot, P. C. and Vanstone, S. A. (1996). *Handbook of applied cryptography*, CRC press.

Kayalvizhi, R., Vijayalakshmi, M. and Vaidehi, V. (2010). "Energy analysis of rsa and elgamal algorithms for wireless sensor networks." In *International Conference on Network Security and Applications*, Springer, 172–180.

Kinnis, T. F. and Sit, H. W. (2005). "Digital signature service.")US Patent 6,959,382.

Kittur, A. S., Jain, A. and Pais, A. R. (2017). "Fast verification of digital signatures in iot." In *International Symposium on Security in Computing and Communication*, Springer, 16–27.

Kittur, A. S., Kauthale, S. and Pais, A. R. (2019). "Bad signature identification in a batch using error detection codes." In *International Conference on Security & Privacy*, Springer, 53–66.

Kittur, A. S. and Pais, A. R. (2017). "Batch verification of digital signatures: Approaches and challenges." *Journal of Information Security and Applications*, 37, 15–27.

Kittur, A. S. and Pais, A. R. (2019a). "A new batch verification scheme for ECDSA* signatures." *Sadhana*, 44(7), 157.

Kittur, A. S. and Pais, A. R. (2019b). "A trust model based batch verification of digital signatures in iot." *Journal of Ambient Intelligence and Humanized Computing*, 1–15.

Koblitz, N. (1987). "Elliptic curve cryptosystems." *Mathematics of computation*, 48(177), 203–209.

Koblitz, N. (1998). "An elliptic curve implementation of the finite field digital signature algorithm." In *Annual International Cryptology Conference*, Springer, 327–337.

Koblitz, N. and Menezes, A. (2005). "Pairing-based cryptography at high security levels." *Lecture notes in computer science*, 3796, 13.

Kocher, P. C. (1996). "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems." In *Annual International Cryptology Conference*, Springer, 104–113.

Kravitz, D. W. (1993). "Digital signature algorithm.")US Patent 5,231,668.

Krawczyk, H., Bellare, M. and Canetti, R. (1997). "Hmac: Keyed-hashing for message authentication." Technical report.

Lee, B., Boyd, C., Dawson, E., Kim, K., Yang, J. and Yoo, S. (2003). "Providing receipt-freeness in mixnet-based voting protocols." In *International Conference on Information Security and Cryptology*, Springer, 245–258.

Li, C.-T., Hwang, M.-S. and Chen, S. (2010). "A batch verifying and detecting the illegal signatures." *International Journal of Innovative Computing, Information and Control*, 6(12), 5311–5320.

Liao, Y.-P. and Hsiao, C.-M. (2014). "A secure ecc-based rfid authentication scheme integrated with id-verifier transfer protocol." *Ad Hoc Networks*, 18, 133–146.

Lim, C. H. and Lee, P. J. (1994). "Security of interactive dsa batch verification." *Electronics letters*, 30(19), 1592–1592.

Lin, C.-H., Hsu, R.-H. and Harn, L. (2005). "Improved dsa variant for batch verification." *Applied mathematics and computation*, 169(1), 75–81.

MacKay, D. J. and Neal, R. M. (1996). "Near shannon limit performance of low density parity check codes." *Electronics letters*, 32(18), 1645–1646.

Manuel, P. (2015). "A trust model of cloud computing based on quality of service." *Annals of Operations Research*, 233(1), 281–292.

Mármol, F. G. and Pérez, G. M. (2011). "Providing trust in wireless sensor networks using a bio-inspired technique." *Telecommunication systems*, 46(2), 163–180.

Miller, V. S. (1985). "Use of elliptic curves in cryptography." In *Conference on the Theory and Application of Cryptographic Techniques*, Springer, 417–426.

Min-Shiang, H., Cheng-Chi, L. and Yuan-Liang, T. (2001). "Two simple batch verifying multiple digital signatures." In *International Conference on Information and Communications Security*, Springer, 233–237.

Mitra, N., Lafon, Y. et al. (2003). "Soap version 1.2 part 0: Primer." *W3C recommendation*, 24, 12.

Mykletun, E., Narasimha, M. and Tsudik, G. (2006). "Authentication and integrity in outsourced databases." *ACM Transactions on Storage (TOS)*, 2(2), 107–138.

Naccache, D., M'RaÏhi, D., Vaudenay, S. and Raphaeli, D. (1994). "Can dsa be improved?—complexity trade-offs with the digital signature standard—." In *Workshop on the Theory and Application of of Cryptographic Techniques*, Springer, 77–85.

Naor, M. and Yung, M. (1989). "Universal one-way hash functions and their cryptographic applications." In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, ACM, 33–43.

Naor, M. and Yung, M. (1990). "Public-key cryptosystems provably secure against chosen ciphertext attacks." In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, ACM, 427–437.

Ndiaye, M., Hancke, G. and Abu-Mahfouz, A. (2017). "Software defined networking for improved wireless sensor network management: A survey." *Sensors*, 17(5), 1031.

Nguyen, P. Q. and Shparlinski, I. E. (2002). "The insecurity of the digital signature algorithm with partially known nonces." *Journal of Cryptology*, 15(3), 151–176.

Nguyen, P. Q. and Stern, J. (2001). "The two faces of lattices in cryptology." In *Cryptography and lattices*, Springer, 146–180.

Pastuszak, J., Michałek, D., Pieprzyk, J. and Seberry, J. (2000a). "Identification of bad signatures in batches." In *International Workshop on Public Key Cryptography*, Springer, 28–45.

Pastuszak, J., Pieprzyk, J. and Seberry, J. (2000b). "Codes identifying bad signatures in batches." In *International Conference on Cryptology in India*, Springer, 143–154.

Peterson, W. W. and Brown, D. T. (1961). "Cyclic codes for error detection." *Proceedings of the IRE*, 49(1), 228–235.

Peukert, D. (1987). *Die Weimarer Republik*, volume 9, VEB Deutscher Verlag für Musik.

Pfitzmann, B. and Pfitzmann, A. (1989). "How to break the direct rsa-implementation of mixes." In *Workshop on the Theory and Application of of Cryptographic Techniques*, Springer, 373–381.

Pham, C. (2014). "Communication performances of ieee 802.15. 4 wireless sensor motes for data-intensive applications: A comparison of waspmote, arduino mega, telosb, micaz and imote2 for image surveillance." *Journal of Network and Computer Applications*, 46, 48–59.

Pi, R. (3). "Model b+-raspberry pi", raspberry pi, 2018.").

Ram, K. S. S. and Gupta, A. (2016). "Iot based data logger system for weather monitoring using wireless sensor networks." *International Journal of Engineering Trends and Technology*, 32(2), 71–75.

Ren, Y., Wang, S., Zhang, X. and Hwang, M.-S. (2015). "An efficient batch verifying scheme for detecting illegal signatures.." *IJ Network Security*, 17(4), 463–470.

Rescorla, E. and Modadugu, N. (2012). "Datagram transport layer security version 1.2." Technical report.

Rivest, R. L., Shamir, A. and Adleman, L. (1978). "A method for obtaining digital signatures and public-key cryptosystems." *Communications of the ACM*, 21(2), 120–126.

Sabater, J. and Sierra, C. (2005). "Review on computational trust and reputation models." *Artificial intelligence review*, 24(1), 33–60.

Selcuk, A. A., Uzun, E. and Pariente, M. R. (2004). "A reputation-based trust management system for p2p networks." In *ccgrid*, IEEE, 251–258.

Serret-Avila, X. and Boccon-Gibod, G. (2004). "Methods and systems for encoding and protecting data using digital signature and watermarking techniques.")US Patent 6,785,815.

Seungwon, L., Seongje, C. and Yookun, C. (2006). "Efficient identification of bad signatures in rsa-type batch signature." *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 89(1), 74–80.

Shao, Z. (2001). "Batch verifying multiple dsa-type digital signatures." *Computer Networks*, 37(3), 383–389.

Shelby, Z., Hartke, K. and Bormann, C. (2014). "The constrained application protocol (coap)." Technical report.

Shen, X., Liu, Z., Harn, L. and Lou, Y. (1999). "A batch-verifying algorithm for multiple digital signatures." *Proc. Of the Iasted, November, MIT, Boston, USA*.

Stallings, W. (2006). *Cryptography and network security: principles and practices*, Pearson Education India.

Suo, H., Wan, J., Zou, C. and Liu, J. (2012). "Security in the internet of things: a review." In *Computer Science and Electronics Engineering (ICCSEE), 2012 International Conference on*, volume 3, IEEE, 648–651.

Sweeney, P. (1991). *Error control coding*, Prentice Hall UK.

Tanner, R. (1981). "A recursive approach to low complexity codes." *IEEE Transactions on information theory*, 27(5), 533–547.

Vaeth, J. S. and Walton, C. S. (2000). "Virtual certificate authority.")US Patent 6,035,402.

Van Arem, B., Tampère, C. and Malone, K. (2003). "Modelling traffic flows with intelligent cars and intelligent roads." In *Intelligent Vehicles Symposium, 2003. Proceedings. IEEE*, IEEE, 456–461.

Vaudenay, S. (2003). "The security of dsa and ecdsa." In *International Workshop on Public Key Cryptography*, Springer, 309–323.

Vu, Q.-A. N., Canal, R., Gaudou, B., Hassas, S. and Armetta, F. (2012). "Trustsets: using trust to detect deceitful agents in a distributed information collecting system." *Journal of Ambient Intelligence and Humanized Computing*, 3(4), 251–263.

Wander, A. S., Gura, N., Eberle, H., Gupta, V. and Shantz, S. C. (2005). "Energy analysis of public-key cryptography for wireless sensor networks." In *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*, IEEE, 324–328.

Wang, F.-Y., Zeng, D. and Yang, L. (2006). "Smart cars on smart roads: an ieee intelligent transportation systems society update." *IEEE Pervasive Computing*, 5(4), 0068–69.

Wang, R., Gu, F. and Shen, E. (2007). "Advances in cognitive neurodynamics." In *the Proceedings of Internatinal Conference on Cognitive Neurodynamics, Springer Publication*, Springer.

Wolf, J. K. and Blakeney, R. D. (1988). "An exact evaluation of the probability of undetected error for certain shortened binary crc codes." In *MILCOM 88, 21st Century Military Communications-What's Possible?'. Conference record. Military Communications Conference*, IEEE, 287–292.

Wu, F., Rüdiger, C. and Yuce, M. R. (2017). "Real-time performance of a self-powered environmental iot sensor network system." *Sensors*, 17(2), 282.

Wu, F., Xu, L., Kumari, S., Li, X., Das, A. K. and Shen, J. (2018). "A lightweight and anonymous rfid tag authentication protocol with cloud assistance for e-healthcare applications." *Journal of Ambient Intelligence and Humanized Computing*, 9(4), 919–930.

Xiong, L. and Liu, L. (2004). "Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities." *IEEE transactions on Knowledge and Data Engineering*, 16(7), 843–857.

Zhang, Z.-K., Cho, M. C. Y., Wang, C.-W., Hsu, C.-W., Chen, C.-K. and Shieh, S. (2014). "Iot security: ongoing challenges and research opportunities." In *2014 IEEE 7th International Conference on Service-Oriented Computing and Applications*, IEEE, 230–234.

Zhiwei, G., Yingxin, H. and Kai, L. (2015). "Cptias: a new fast pki authentication scheme based on certificate path trust index." *Journal of Ambient Intelligence and Humanized Computing*, 6(6), 721–731.

Zhou, R. and Hwang, K. (2007). "Powertrust: A robust and scalable reputation system for trusted peer-to-peer computing." *IEEE Transactions on Parallel & Distributed Systems*, (4), 460–473.

Zhu, Q., Wang, R., Chen, Q., Liu, Y. and Qin, W. (2010). "Iot gateway: Bridgingwireless sensor networks into internet of things." In *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on*, IEEE, 347–352.

PUBLICATIONS BASED ON THE RESEARCH WORK

1. Kittur, A. S., and Pais, A. R. (2017). Batch verification of Digital Signatures: Approaches and challenges. *Journal of Information Security and Applications (Elsevier)*, 37: 15–27. **[Published]**

2. Kittur, A. S., and Pais, A. R. (2019). A Trust Model based Batch Verification of Digital Signatures in IoT. *Journal of Ambient Intelligence and Humanized Computing (Springer)*, 1–15 **[Published]**

3. Kittur, A. S., and Pais, A. R. (2019). A New Batch Verification Scheme for ECDSA* Signatures. *Sādhanā (Springer)*, 44(7): 157. **[Published]**

4. Kittur, A. S., Jain, A., and Pais, A. R. (2017, September). Fast Verification of Digital Signatures in IoT. *International Symposium on Security in Computing and Communication* (pp. 16-27). Springer, Singapore. **[Published]**

5. Kittur, A. S., Kauthale, S., and Pais, A. R. (2018). Bad Signature Identification in a Batch using Error Detection Codes. *International Conference on Security and Privacy 2019 (ISEA-ISAP 2019) (pp. 53-66) (Springer)* Singapore **[Published]**

BIO-DATA

Name:	Apurva S. Kittur
Date of Birth:	12/03/1991
Gender:	Female
Marital Status:	Married
Father's Name:	Dr. Shreekant Kittur
Mother's Name:	Sneha Kittur
Husband's Name:	Mr. Raghunath Kulkarni
Email Id:	apurva.kittur@gmail.com
Present Address:	SB G1, Kumardhara Block, Vijaya Enclave, Bilekhalli, Bannerghatta Road, Bangalore, Karnataka, INDIA-560076
Educational Qualifications:	B.Tech (CSE) - KLS Gogte Institute of Technology, Belagavi, Karnataka
	M.Tech (Networking and Internet Engineering) - Sri Jayachamarajendra College of Engineering, Mysore, Karnataka
Areas of Interest:	Security in IoT, Cryptography, High Performance Computing, Information Security