2005

Efficient trapdoor-based client puzzle system against DoS attacks

Yi Gao
University of Wollongong, yigao@uow.edu.au

Efficient Trapdoor-Based Client Puzzle System Against DoS Attacks

A thesis submitted in fulfillment of the
requirements for the award of the degree

Master of Computer Science by Research

from

UNIVERSITY OF WOLLONGONG

by

Yi Gao

School of Information Technology and Computer Science
October 2005

Dedicated to my parents:
Qizhai Gao and Shujie Sun

Declaration

I, Yi GAO, declare that this thesis, submitted in partial fulfilment of the requirements for the award of Master of Computer Science by research, in the School of Information Technology and Computer Science, University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. The document has not been submitted for qualifications at any other academic institution.

Yi Gao
October 24, 2005

Publication

Chapter in Book

Yi Gao, Willy Susilo, Yi Mu, and Jennifer Seberry. "Efficient Trapdoor Based Client Puzzle Against DoS Attacks." *Network Security*, S. Huang, D. MacCallum and D-Z. Du (eds.), Springer-verlag, 2005 (to appear).

Conference Paper

Yi Gao, Yi Mu and Willy Susilo. "Preventing DoS Attacks with A New Client Puzzle Scheme." *In Proceedings of the AUUG'2005 Annual Conference*, October 2005 (to appear).

Abstract

Denial of service (DoS) and distributed denial of service (DDoS) are serious threats to computer networks. DoS and DDoS attacks aim to shut down a target server by depleting its resources and rendering it incapable of offering stable and integrated service to legitimate clients. Preventing DoS and DDoS attacks is a difficult task. A promising countermeasure against DoS attacks is the *Client Puzzle* method, which nevertheless faces a number of challenges, such as the complexity of puzzle construction and solution verification.

Our research focuses on exploring novel puzzle constructions to satisfy the high demands of DoS defence in practice. In this thesis, we first identify the underlying weaknesses of existing client puzzles. To mitigate these vulnerabilities, we recommend the necessary requirements for good client puzzles. Based on this, we propose a new model for puzzle distribution, called the Trapdoor-based Client Puzzle System (TCPS). Two specific schemes are presented to construct puzzles within TCPS. We depict these two schemes, where each trapdoor algorithm is applied respectively. Both schemes have two distinct features: the computational overheads are low, and the difficulty level of puzzles is measurable. Moreover, both puzzle schemes are provably secure under traditional hard problems in mathematics.

Our contribution to client puzzle defence against DoS attacks can be summarised as follows:

- Identify the shortcomings of existing client puzzles.
- Recommend the requirements of good client puzzles.
- Formally define the Trapdoor-based Client Puzzle System, along with strict security conditions.
- Propose a client puzzle scheme whose security is based on the RSA Assumption. Effectiveness and security are analysed and proven.

- Propose a second client puzzle scheme whose security is based on the Discrete Logarithm Problem (DLP). Similarly, effectiveness and security are also analysed.
- Provide a possible configuration for system parameters.
- Discuss further possible attacks and their solutions.

As our research is carried out in DoS attack scenarios, we also introduce this technical background before our achievements are presented.

Acknowledgements

The research work for this thesis was undertaken at the University of Wollongong.

First, I would like to thank my supervisors, Associate professor Yi Mu and Dr. Willy Susilo for their guidance and help in my research. Without them, this thesis would not have been possible.

I greatly appreciate Ruth Walker and Juliet Richardson for their kind help in the English expression of this thesis. I also wish to acknowledge the support I have received from all the staff in the School of IT & CS, University of Wollongong.

Finally, I would like to thank my family and friends for their enduring love and support.

Contents

Publication v

Abstract vi

Acknowledgements viii

1 Introduction 1

1.1 Service Availability 1

1.2 Overview of DoS Attacks 2

1.3 Previous Work 3

1.4 The Challenge 5

1.5 Thesis Organisation 5

2 Technical Background 7

2.1 A Brief History of DoS Attacks 7

2.2 Underlying Causes of DoS Attacks 9

2.2.1 Social Factors 9

2.2.2 Architectural Factors 9

2.3 Classification of DoS Attacks 10

2.4 Representative DoS Attacks 12

2.4.1 IP Spoofing 12

2.4.2 TCP SYN Flooding 13

2.4.3 Smurf 15

2.4.4 DDoS 16

2.5 Proposed Countermeasures 22

2.5.1 Ingress/Egress Filtering . . . 22
2.5.2 Packet Marking . . . 24
2.5.3 Time-out . . . 24
2.5.4 Random Dropping . . . 25
2.5.5 SYN Cookies . . . 25
2.6 Summary . . . 26

3 Client Puzzles **27**
3.1 Overview of Client Puzzles . . . 27
3.2 Application of Puzzles . . . 29
3.2.1 Puzzles in Key Agreement . . . 29
3.2.2 Puzzles in Junk Mail Defence . . . 30
3.3 Client Puzzle Protocol . . . 31
3.4 Puzzle Construction . . . 33
3.4.1 Hash Function-based Puzzle Scheme . . . 34
3.4.2 Diffie-Hellman Based Puzzle Scheme . . . 36
3.5 Summary . . . 39

4 Trapdoor-based Client Puzzle System **40**
4.1 Problems . . . 40
4.2 Essential Requirements . . . 41
4.3 Definition . . . 42
4.3.1 One-Way Function . . . 43
4.3.2 Trapdoor One-Way Function . . . 43
4.3.3 Trapdoor-based Client Puzzle System . . . 43
4.4 Security Assumption . . . 46
4.4.1 The RSA Assumption . . . 47
4.4.2 The Discrete Logarithm Problem (DLP) . . . 48

5 A Scheme Based On The RSA Assumption **49**
5.1 Algorithm . . . 49
5.2 A RSA Assumption-based TCPS . . . 52

5.3 Parameter Table and Scheme Prototype 57
5.4 Remarks 59
5.5 Security Considerations 62
5.6 Summary 63

6 A Scheme Based On The DLP **65**
6.1 Algorithm 65
6.2 A DLP-based TCPS 68
6.3 Parameter Table and Scheme Prototype 72
6.4 Remarks 74
6.5 Security Considerations 76
6.6 Summary 77

7 System Description and Discussion **78**
7.1 Working Environment 78
7.2 System Description 79
7.3 Discussion 81

8 Conclusion and Future Work **85**

Bibliography **89**

List of Tables

1.1 General Security Goals and Threats 1

2.1 DoS Classifications 12

4.1 The Relationship Between Factorisation, RSA Assumption and RSA . . 48

5.1 Parameter Table of the RSA Assumption-based TCPS 57

6.1 Parameter Table of the DLP-based TCPS 72

List of Figures

2.1 IP Spoofing Attack 13
2.2 TCP's Three-way Handshake 14
2.3 TCP SYN Flooding Attack 15
2.4 Smurf Attack 17
2.5 Three-layer Control For A DDoS Attack 18
2.6 Reflection DDoS Attack 21
2.7 Ingress Filtering 23

3.1 No DoS Attack Threat 32
3.2 Under A DoS Attack 33
3.3 Aura's Client Puzzle Construction 34
3.4 D-H-based Client Puzzle Construction 37

4.1 A Simple Prototype For The TCPS 45

5.1 A Sample of Set A 54
5.2 A RSA Assumption-based Puzzle Scheme 58

6.1 A DLP-based Puzzle Scheme 73

7.1 A Threat To Our TCPS 81
7.2 Using A Sequence Number Against IP Spoofing 83

Chapter 1

Introduction

1.1 Service Availability

Today's Internet has successfully fulfilled the expectation of empowering a single computer to service remote requests from millions of geographically dispersed clients. With this significant power, the Internet has been widely applied in our society, and has increasingly become a prevalent part of human lives. People enjoy and benefit greatly from a number of fresh nouns that belong to a new information era: eBusiness, eCommerce, eEducation, eGovernment, eHealth and so on. In consequence, the issue of how to supply these network services reliably and securely to legitimate clients is a growing concern among network engineers and researchers.

Generally, authentication, integrity and confidentiality are the most important principles of network security. However, recent reports about a number of prominent Internet service providers that broke down because of malicious attacks [2, 3, 66, 68] urge people to realise that all security principles must be based on service availability. It is clear that no one can evaluate the quality of an online service that is not available. "Availability" in this context refers to a service that can be accessed within a reasonable amount of waiting time after a legitimate client sends a request. Table (1.1) illustrates these general security goals and their corresponding threats.

Goal	**Security Threat**
Information Confidentiality	Exposure of Information
Information Integrity	Modifying/Injecting Information
Information Authentication	Forged Information
Service Availability	Denial of Service

Table 1.1: General Security Goals and Threats

The service availability of a network server can be destructed in a variety of ways, such as internal bugs within a system, hardware limits, or malicious attacks from outside. *Denial of Service* (DoS) is the term we use to refer to the results of any intentional or accidental actions that can successfully make a legitimate service unavailable for legitimate users.

Since other DoS threats are relatively easy to deal with, this thesis will focus only on the study of malicious DoS attacks launched via the Internet. By analysing existing countermeasures against DoS attacks and their inherent problems, we will propose a new defence scheme, which is expected to be efficient and effective in both practice and theory.

1.2 Overview of DoS Attacks

D. Howard [32] presented a comprehensive definition of DoS:

> *If computer hardware, software, and data are not kept available, productivity can be degraded, even if nothing has been damaged. Denial of Service can be conceived to include both intentional and unintentional assaults on a system's availability. The most comprehensive perspective would be that regardless of the cause, if a service is supposed to be available and it is not, then service has been denied.*

Network-based DoS attacks, in particular, denote malicious actions which aim at shutting down a target server and destructing its service availability. These attacks usually attempt to block or degrade service in a designated period temporarily, rather than intrude on the server directly or damage the data permanently. Similar attack scenarios can be found in the real world: a malicious client repeatedly uses distinct bogus names, phone numbers and credit card numbers to deceive a hotel receptionist into reserving rooms for him, which as a result, are not available for other legitimate clients in a certain period. This is a typical DoS attack.

A Cyberworld DoS attack is designed to flood numerous spurious requests to a server, crushing its infrastructure, depleting its bandwidth, computational capacity or system stack memory, and eventually crashing it. In consequence, legitimate clients have to experience a service downtime, and victim servers may lose millions of dollars.

One of the most popular DoS attacks, TCP SYN flooding attacks, were reported by several major newspapers in 1996 [2, 3, 20, 68]. These attacks succeeded in crippling

Panix, a major New York Internet service provider, in early September 1996, and created similar problems for the website of the New York Times just a few days later. As a rule, a SYN flooding attacker exploits spoofed IP addresses to mount a large number of initial and unresolved connection requests to a victim server, depleting its resources and rendering it incapable of responding to legitimate clients. According to an advisory issued by CERT on September 19th, 1996 [59], several underground magazines pushed the spread of DoS attacks ahead by publishing source codes and automated tools for launching TCP SYN flooding attacks.

Distributed Denial of Service (DDoS) was rapidly brought to the public's attention after eBay, Amazon, Yahoo and several other prominent commercial websites fell victim to this new form of DoS attacks on February 9th, 2000 [16, 66]. Relying on the fast spread of Internet worms [15], a DDoS attacker is able to easily manipulate thousands of vulnerable computers in the Internet to launch a large-scale DoS attack to a target. Compared with traditional ones, the strength of DDoS attacks can be multiplied by 10, 100, or even 1000, and the effect on the Internet is therefore immeasurable.

In a worse case scenario, as the above incidents have demonstrated, even if an Internet server possesses very large bandwidth and resources, and is protected by a reliable firewall system, it is still vulnerable to a range of Denial of Service attacks.

1.3 Previous Work

A countermeasure against bogus clients is to perform authentication before any communication and resource allocation. However, conventional authentication schemes based on public-key technology (for example, RSA and DSS) will no longer meet the demand for DoS defence. This is because most of them require a server to conduct expensive computations, such as modular exponentiation, and store a large amount of session information for each client, which actually opens up new opportunities for DoS attacks. Therefore, a more effective and inexpensive approach to defend against DoS attacks is desirable.

So far, several approaches have been proposed in the literature [34, 33, 44, 43], among which *Client Puzzle* is one of the most notable and influential. Earlier work [7, 5, 33] showed that the client puzzle mechanism is capable of alleviating or confining DoS attacks to a harmless level in theory. The aim of client puzzles is to destroy DoS attacks by forcing every suspected adversary to consume a number of computational resources for authentication, before he/she is granted access to the resources of a server.

In contrast to traditional authentications, client puzzles seem weak, yet are inexpensive and efficient in determining whether a connection request is sent by network worms [15, 63].

The idea of client puzzles was first introduced as an access control mechanism by Dwork and Naor in 1992 [22]. They proposed a system for junk mail defence, in which every successful delivery of a message requires the sender to solve a small cryptographic puzzle. By doing so, they successfully impose a large amount of computational costs on sending mass mails, while for legitimate clients, the costs to compute single puzzles are negligible.

Combining the idea of a stateless protocol [4] and *Client Puzzle*, Juels and Brainard [33] proposed a *Client Puzzle Protocol* to protect network servers against SYN flooding attacks. This protocol emphasises that no memory should be allocated before client authentication, and that the client is the one who pays for the authentication. Generally speaking, when there is no DoS attack alarm, a defending server accepts and responds to connection requests as normal. However, if the server is suspected of being under attacks, it will send a small cryptographic puzzle to each client applying for a service, before allocating any system resources to them. Only the request belonging to the client who returns the correct answer will proceed. The cost of computing a single puzzle is trivial for legitimate clients, yet unbearably expensive for a DoS attacker who attempts to consume considerable resources from the server. Moreover, the complexity of cryptographic puzzles can be adjusted by an administrator, according to the strength of the attack he/she receives. In their paper, Juels and Brainard also presented a simple puzzle construction to implement their protocol, although this seemed unsatisfactory and caused a lot of arguments in network forums [19, 48, 65].

Following this, a few researchers attempted to improve puzzle construction within the framework of the *Client Puzzle Protocol*. Aura and Nikander [5] proposed a hash function based puzzle scheme, in which a client needs a brute-force search for the correct answer, and a server performs a hash function to verify the solution. Waters and Juels [6] suggested a new technique that permits the outsourcing of puzzles. However, even puzzles can be used by different servers, and the solution of a puzzle still requires one modular exponentiation for every defending server.

In general, these proposed puzzle schemes are less capable of meeting the requirements of client puzzles. Improper usage of these unqualified puzzles, on the other hand, will lead to DoS attacks.

1.4 The Challenge

The aim of this study is to redefine the essential requirements of client puzzles, and to establish a generic client puzzle system which is appropriate and secure enough to be embedded in the *Client Puzzle Protocol* [33].

It is desirable that this system should possess two prominent characteristics. One is that most of the computations for puzzle generation can be fulfilled in a pre-construction phase, independent of a puzzle construction. Pre-construction can be processed during idle time, and items calculated in this phase can be reused by combining them with time parameters. The other important feature is a quick verification. No computation occurs in the verification phase, which makes DoS attacks aiming at flooding bogus solutions to exhaust system resources impossible. Moreover, the complexity of puzzles in the system proposed in this thesis are parameterised and can be adjusted flexibly, according to the strength of DoS attacks.

We will propose two novel trapdoor algorithms to implement this client puzzle system. Both of them are provably secure under the assumptions of well-known hard problems such as Discrete Logarithm Problem and Factorisation [36, 50]. We will demonstrate how a trapdoor puzzle can be generated by a defending server, and how a client achieves the correct answer. By counting their respective workloads, the advantages of our scheme will be analysed in contrast with other proposed puzzles.

Our scheme is expected to overcome the potential problems of previous client puzzles, such as complicated construction and verification. We will demonstrate that our puzzles are computationally efficient and properly cohere with existing Internet protocols.

1.5 Thesis Organisation

The rest of the thesis is organised as follows:

- In Chapter 2, we will present the developing history of DoS attacks, and analyse potential reasons for their existence. We will give a broad overview of DoS attacks and their attack modes by depicting a number of prevalent examples, such as SYN flooding, Smurf, and DDoS. Several proposed countermeasures will be described in this chapter, along with an analysis of their advantages and disadvantages.

- Chapter 3 will be devoted to the notion of client puzzles. Three relevant applications of cryptographic puzzles (including the Client Puzzle Protocol [33]) will be addressed in this chapter. To clarify the problems we try to resolve in this thesis, two proposed client puzzle schemes will be investigated. Note that Chapter 3 and Chapter 4 are intended to provide the background knowledge needed for the following chapters.

- To solve current problems and enable client puzzles to prevent DoS attacks effectively, we will propose a novel model for puzzle construction, called the Trapdoor-based Client Puzzle System (TCPS). This model will consist of our new recommendations about qualified client puzzles, and a series of security conditions. It is hoped that puzzles generated from our proposed model will fulfill their original promise against DoS attacks.

- We will develop two specific trapdoor algorithms and describe them in Chapter 5 and Chapter 6. Each individual algorithm can be used to implement an efficient puzzle construction within our proposed model. We will demonstrate how the TCPS work by employing these two algorithms, respectively. We will provide rigorous proof of their security properties. Moreover, a number of possible attacks will be considered at the end of both chapters, and the proposed puzzle schemes are expected to resist all of them.

- In Chapter 7, we will provide the proper parameterisations for a defending system, which will help our puzzle schemes work effectively in practice. Some discussion about further attacks and security considerations will be presented.

- Chapter 8 is the conclusion, where we will summarise our work within this thesis, together with a list about future research directions.

- The last part of this thesis comprises the references.

Chapter 2

Technical Background

Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS) attacks have been around for years, but it seems that people still cannot find a proper way to deal with them. This chapter intends to provide an overview of DoS/DDoS attacks, along with several proposed countermeasures, which, it is hoped, will help readers understand the problems that we try to solve in the following chapters.

This chapter will begin with a brief history of DoS attacks, which leads to a review of their evolution. A discussion about why DoS attacks exist and are rapidly developing follows. DoS attacks will be classified according to their characteristics, and several notorious DoS attack approaches will also be described in this chapter. After that, a number of current countermeasures against DoS attacks will be introduced, such as Packet Marking, Filtering techniques, SYN cookies, and so on. These countermeasures' objectives, working theories, merits and limitations will also be analysed.

2.1 A Brief History of DoS Attacks

Most people were not aware of the urgency of DoS attacks, until a number of famous Internet service providers were brought down in February 1996. However, earlier than this, in 1988, an incident happened which was ignored, maybe because the Internet was relatively unknown at that time. The Morris worm, a self-replicating program created by Robert T. Morris Jr., successfully disrupted the Internet for nearly 48 hours in the city of California [53]. That was the first taste of DoS attacks.

After ten years of development, DoS attacks appear more sophisticated and difficult to solve. Before 1999, DoS attackers might have exploited IP spoofing and flaws in existing network protocols to mount attacks from a single source to a single target. A

series of advisories issued by CERT [59, 56, 60, 57, 61] between 1996 and 1998 reported these attacks, such as SYN flooding, Smurf, ping of death, etc.

In 1999, several Distributed Denial-of-Service tools (Trinoo, TFN and "stacheldraht") were reported by CERT [58] for the first time. The fear about larger-scale attacks proved to be true by the events that happened in February, 2000, when eminent websites like Yahoo, CNN.com and Amazon, which were protected by powerful firewall systems and possessed huge bandwidth and system resources, were still shut down by DDoS attacks.

The most common process for launching DDoS attacks consists of three steps:

1. Scan vulnerable computers across a wide range of the Internet.
2. Intrude on these victims and install malicious scripts for mounting an attack. According to different types of scripts, these infected computers are categorised as "Masters" or "Zombies".
3. An attacker communicates with "Masters" only, deploying them to transfer an attack order to "Zombies", which will finally mount the real attack.

Before 2000, most DDoS attacks were required to scan vulnerable victims manually, and list them for later intrusion. The attacks took the form of multiple sources to a single target at this stage.

From 1996 to 2000, several countermeasures against DoS/DDoS were proposed, such as SYN cookies [8], Filtering mechanisms [25, 43], Congestion Control [41, 55], etc. However, none of them seemed good enough to tackle and stop the violence of DoS attacks.

Since 2001, the quick evolution of DDoS attacks is even more terrifying. Attackers who deploy self-propagating network worms like the notorious Code Red and advanced scanning strategies [63] can easily compromise more than ten thousand unwitting "accomplices" in a few hours. Stefan Savage, a network researcher from CAIDA (Supercomputer Center's Cooperative Association for Internet Data Analysis) pointed out that, "With that kind of firepower, they could have taken down anything" [31]. Scientists admit that with a little improvement, Code Red could render an arbitrary network incapable of communicating with the outside world. This is a new phase of DoS attacks which can be launched from multiple sources to multiple targets.

Furthermore, communication between "Zombies" and "Masters" can be encrypted by DoS tools such as "stacheldraht", and be transferred via IRC channels [14] which

offer anonymous service for their users. All of these factors make it more difficult to detect and trace a DDoS attack.

2.2 Underlying Causes of DoS Attacks

2.2.1 Social Factors

Everything exists for a reason. If the first appearance of the Morris worm was due to intellectual curiosity [53], the prevalence of DoS attacks nowadays has a much more realistic basis in this for-profit society.

First of all, business or commerce being performed via the Internet means high profit and time sensitiveness [28]. A successful DoS attack may lead to a victim losing thousands of dollars per minute. Hence, a DoS attack may aim to commit a commercial crime or take personal revenge. In addition, the purpose of bringing down popular Web servers can also be to gain a reputation among hackers' community.

More evidence indicates that the tools to automatically launch DoS attacks that disperse within the Internet have become another serious problem. Even a network illiterate can mount a considerable DoS attack by using these tools. This situation is not acceptable in today's Internet, where numerous monetary transactions are handled. What we need is not only a good solution to defend against these DoS attack tools, but also a complete policy to rule the activity of information providers in the Internet. Unfortunately, this idea seems far from actual practice and difficult to achieve.

2.2.2 Architectural Factors

The Internet was created for functionality, not for security. It can supply worldwide clients with quick, easy and inexpensive communication channels, and can be gradually reinforced by diverse levels of network protocols that ensure the reliability and timely delivery of communication. However, the booming growth of the Internet also leaves a lot of concealed damage and other serious issues for security researchers.

One problem of the Internet is that network resource is limited and consumable [14]. Bandwidth, processing power or memory of a network device all have their maximum capabilities. When a network computer provides service to a mass of remote clients, it simultaneously creates the possibility of making a single computer fight against a large number of network resources. If the goal of an attack is to deplete the victim's

resources, this can always be achieved in theory, as long as it carries with a sufficient amount of resources, such as a large number of bogus connection requests. Actually, this is why DDoS attacks are successful.

Another problem is that "Internet security is highly interdependent" [14]. For example, DDoS attacks generally launch from systems or networks that are undermined through security-related compromises. That means, no matter how perfectly the target system might be protected, its susceptibility to DDoS attacks depends on the security status of the rest of the global Internet.

The last but not the least issue is the hasty deployment of network protocols. Most of them are designed to meet the demands of industry, and are hastily applied to widespread network servers and routers. SYN Flooding and IP spoofing attacks aim at the shortcomings of TCP/IP [46, 47]. A server based on SSL protocol [27] is subject to DoS attacks, because the protocol requires the server to perform a computationally expensive verification operation to initiate a SSL connection. An attacker may easily overwhelm the server by flooding it with invalid connection requests.

2.3 Classification of DoS Attacks

Before discussing details of specific examples of DoS attacks, it is useful to classify DoS attacks according to their characteristics. Some researchers have undertaken this work in distinct ways. Readers can refer to [39, 34, 67] for more information. Here, we provide two kinds of classification.

1. One possible classification of DoS attacks according to the aims of attacks could be:

   - **System destruction**: The target of this type of attack is the hardware of network devices, such as electricity power, network lines, and so on. These attacks are easy to detect, and can be solved quickly by switching on the backup power system, or recovering the communication lines. Besides these physical attacks, the limitations of hardware, such as a Network card or CPU with too small capability, is also an attack point in this category.

   - **Implementation bugs**: Sometimes attackers may search for specific bugs in network systems, or scan improper configurations of firewalls or routers to launch their attacks. These system and application faults may be caused by the ignorance of administrators, or software bugs. The general solution is to install

patches timely, and examine important input and output data for network devices regularly and patiently.

- **Resource consumption**: A service provided by a server can be viewed as a shared resource in the Internet. A DoS attacker who aims at resource consumption can exploit bogus requests to deceive the victim server into repeatedly granting the resources to him, until it is exhausted or unavailable for other legitimate clients. This type of attack is more difficult to tackle, because most of them make use of the weaknesses of existing network protocols. In fact, their malicious requests appear no different from legitimate ones, and ordinary defence systems are incapable of detecting them.

  According to the definition of resource, forms of attack can be further divided into distinct parts as follows:

  - **System resource**: This includes CPU processing capability, storage capability, buffer space, etc. Attackers cripple a victim by forcing it to process more than it can handle. Notice that this result is not due to the low capability of the hardware. In fact, attackers take advantage of the flaws of several protocols to unlimitedly amplify the effect of DoS attacks. As we discuss later, TCP SYN flooding attacks belong just to this category.
  - **Bandwidth**: When installing a network device, such as a server, a router or a firewall, the administrator will configure the maximum bandwidth or maximum connectivity. The aim of these attackers is to force a server to deplete its connectivity or obstruct the network by flooding large amounts of traffic packets. A Smurf attack forces routers to stop forwarding packets due to network congestion.

2. The other classification relies on the evolution of DoS attacks, and three attack modes can be identified:

- **Single-to-Single**: Early DoS attacks, such as IP Spoofing attacks and TCP SYN flooding attacks, belong to this category. They exploit spoofed IP addresses to cheat a victim server out of resources. But actually, they only use their own system resources to perform these attacks. As a result, the power and impact of these attacks are relatively impotent. Furthermore, they are less available to launch attacks that aim at bandwidth consumption.

- **Multiple-to-Single**: Early Distributed DoS attacks take advantage of network worms to compromise vulnerable machines on the Internet. These raw recruits are used to launch a cooperative DoS attack. In combination with the attack methods used in Single-to-Single, many systems with less resources can attack a much larger system.
- **Multiple-to-Multiple**: Nowadays, more sophisticated DDoS attacks can easily overwhelm a target network that may include several network servers. Although several countermeasures and DoS Detect systems have been applied to mitigate the force of large-scale DDoS attacks, the data on DoS attacks each week indicate that they are still far from being prevented.

The following table summarises the above classifications:

DoS Classification	by Aims:	(1).	Physical destruction
		(2).	Implementation Bugs
		(3).	Resource consumption
	by Modes:	(1).	Single-to-Single
		(2).	Multiple-to-Single
		(3).	Multiple-to-Multiple

Table 2.1: DoS Classifications

2.4 Representative DoS Attacks

2.4.1 IP Spoofing

IP spoofing means to cheat others by using false IP addresses instead of one's own. Strictly speaking, IP Spoofing alone is not a DoS attack, but an important step in those attacks. Nearly all successful DoS attacks need to cooperate with this technique in order to conceal attackers' real IP addresses and avoid IP tracing.

At the beginning, IP spoofing was used by attackers to gain unauthorized access to remote systems. In 1995, one year before the appearance of SYN flooding attacks, CERT reported several IP spoofing attacks [62], in which attackers could obtain root

access to victim systems by making use of applications that used authentication based on IP addresses.

When a client wants to establish a TCP connection, a program can be used to generate a socket, automatically filling the header field of an IP packet with the source address. However, an attacker may modify this program under Unix, and change the address to whatever he wants. Due to the fact that the routing strategy only considers IP destination addresses, the correctness of IP source addresses is unfortunately ignored [30].

In some cases, an attacker can use IP spoofing to launch a small-scale DoS attack, as in the following scenario, albeit that the strength of this attack is limited. An attacker $\mathcal{A}$ forges $\mathcal{B}$'s IP source address and sends a lot of packets to different destinations. As a rule, all the returning IP packets flow to $\mathcal{B}$, which may lead to network congestion.

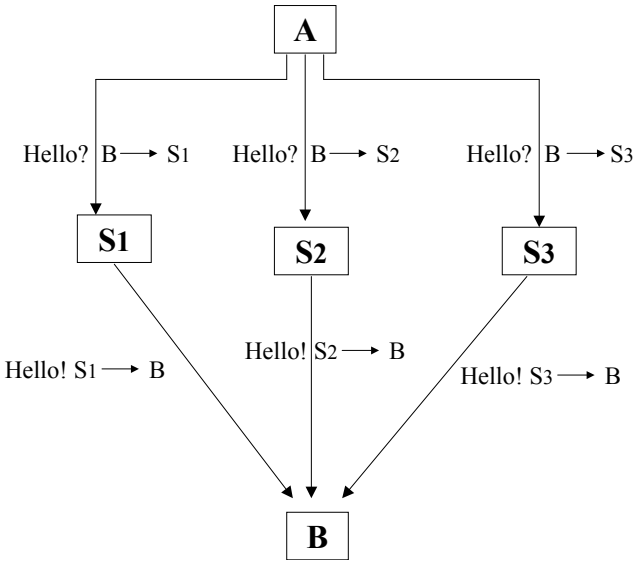

Figure 2.1: IP Spoofing Attack

2.4.2 TCP SYN Flooding

Taking advantage of the imperfections of the TCP connection establishment protocol [33], attackers launch TCP SYN flooding attacks by flooding a target server with many "half-open" connections, which leads to the victim's connection capability being depleted, so that it becomes unavailable for other legitimate clients.

Normally, establishment of a TCP connection requires that both sides communicate: a client and a server exchange an orderly sequence of messages [59]. This process is commonly called TCP's three-way handshake. The client begins this protocol by sending a SYN message to the server, which is listening to connection requests from the network. Acknowledging the SYN message, the server returns a SYN-ACK message to the client, and meanwhile prepares for this connection by distributing a piece of buffer space to store session information. The client completes the protocol by replying to an ACK message. Now, the connection is established, and the service-specific data can be transferred between the client and the server. Fig (2.2) illustrates this three-way handshake connection.

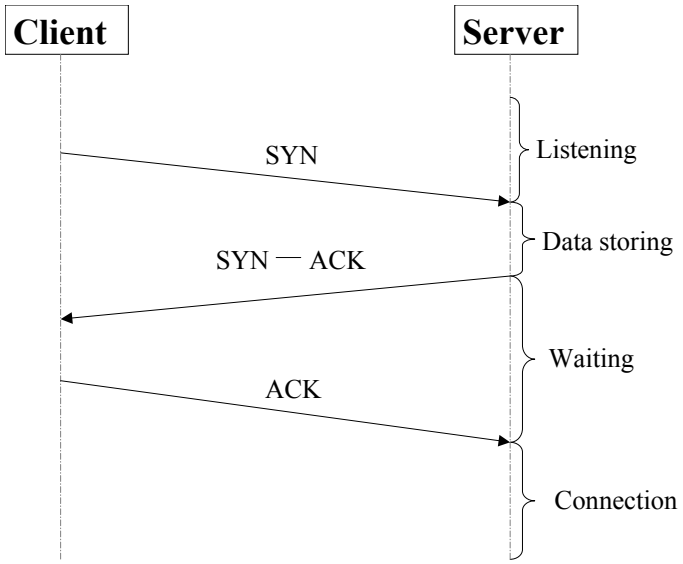

Figure 2.2: TCP's Three-way Handshake

There are at least two shortcomings in this TCP connection establishment protocol. One is that the authentication is based on source IP addresses, whereby an attacker may perform IP spoofing. The other is located at the second step of the three-way handshake, where, before receiving the final ACK message, the server has already allocated buffer space for this connection, and must keep it until a timeout. Taking advantage of this point, an attacker can exhaust the buffer space of the target server by sending a sufficient number of "half-open" connections.

In this attack, the attacker uses IP spoofing to forge large amounts of initial connection requests (SYN messages), and mount them to a target server. These requests

appear to be legitimate and are not filtered by firewall or other defence systems. The server responds with SYN-ACK messages, and allocates buffer space for each connection. Although the time for keeping these reserved buffer space is short, the space can eventually be exhausted, and the server then fails to respond to other legitimate clients, as long as the attacker floods numerous connection requests repeatedly. This attack effectively prohibits normal clients from visiting the target server. That is the reason for its prevalence in the hackers' community.

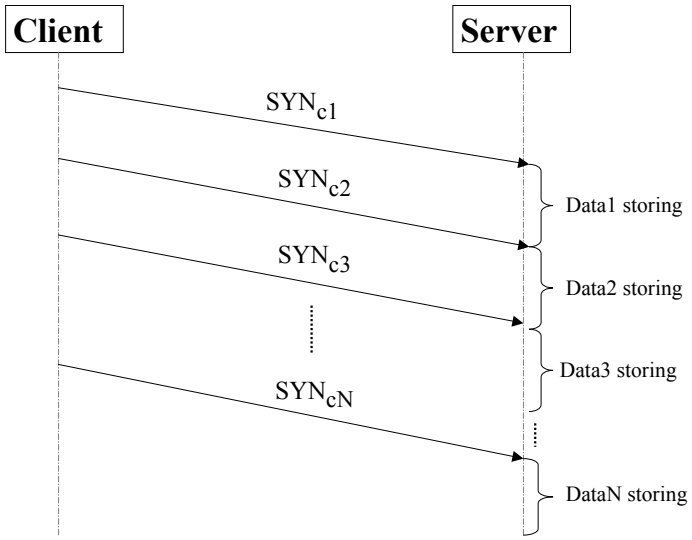

Figure 2.3: TCP SYN Flooding Attack

According to the advisory of CERT [59], any system connected to the Internet and providing a TCP-based network service (such as Web Server, FTP server, or mail server) is potentially vulnerable to this kind of attack.

2.4.3 Smurf

Smurf attacks make use of forged ICMP echo request packets and IP broadcast addresses to overwhelm a victim system with large amounts of ICMP echo reply packets that are sent from an intermediary site [61].

The Internet Control Message Protocol (ICMP) is used to inspect errors and send control messages. It is also used to check whether a network device is responding. Hence, if a machine receives an ICMP echo request packet, it will respond with an ICMP echo reply packet.

On the Internet, a packet can be transferred to an individual IP address or broadcast to an entire subnet, depending on whether the destination address is an IP broadcast address[1]. Via an IP broadcast address, a packet can be delivered to all machines on that subnet.

In a typical Smurf attack, three parties play different roles: an attacker, an intermediary and a victim. Using IP spoofing, the attacker forges an ICMP echo request packet with the victim's source address, and sends it to an IP broadcast address. When all the machines in the intermediary's network receive this packet, they send ICMP echo reply packets directly to the spoofed IP address, which actually belongs to the victim. This may cause severe network congestion in the victim's local network. The targets of this attack may include not only the victim host, but also routers and communication lines connected to the victim's local network. The function of the intermediary site is to amplify the amount of traffic that flows to the victim's address. In consequence, some researchers refer to this site as an amplifier site [34].

Tools for launching this type of DoS attack have been developed [61], which can spread these attacks to multiple intermediaries simultaneously, and lead to much larger attacks. In other cases, the target of Smurf attacks can be the intermediary directly. If all the machines on that network respond to one or several ICMP echo requests, it will certainly cause severe network congestion and outage. In particular, if an attacker can force routers to stop forwarding packets, then all the hosts behind those routers are effectively disconnected.

2.4.4 DDoS

Distributed Denial of Service (DDoS) is a new form of DoS attack, first reported in early 2000 [16, 54, 66]. In contrast to traditional DoS attacks, DDoS attackers, in particular, are armed with self-propagation worms which can be installed on a discretionary number of vulnerable computers on the Internet. An attacker is able to harness these compromised machines in order to mount a coordinated DoS attack. These infected machines are typically divided into two groups: "Masters" and "Zombies", which play different roles in a DDoS attack. "Masters" are more like an intermediary, while "Zombies" serve as attack platforms. Communication between an attacker and the "Zombies" is not direct, but depends on the "Masters". One "Master" may control and deliver the attacker's command to a number of "Zombies". By mounting such a

[1]Each LAN (Local-area Network) on the Internet possess an IP broadcast address.

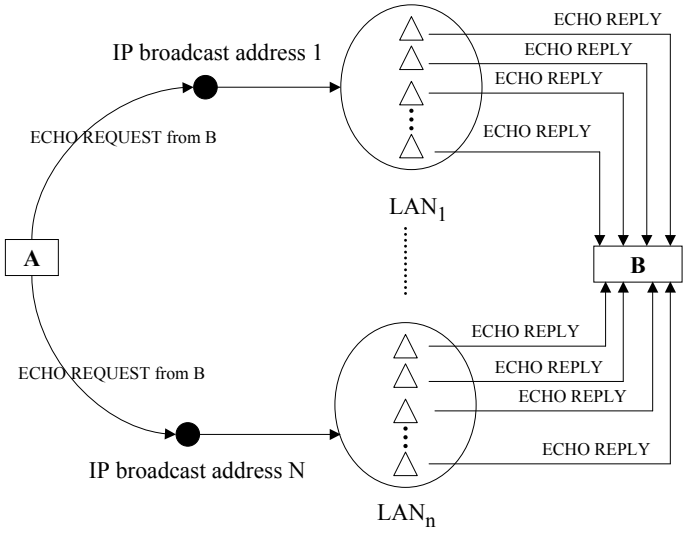

Figure 2.4: Smurf Attack

coordinated DoS attack, the effectiveness of a DDoS can be multiplied by 10, 100, or even 10,000 times [24].

A typical DDoS attack process can be described as follows. An attacker first scans a large range of networks to find vulnerable hosts that have weak defences against a malicious intrusion. The number of these hosts is determined by the strength of the attack that an attacker intends to launch. Second, the attacker installs "Master" or "Agent" programs on these vulnerable hosts. A machine with an "Agent" program is called a "Zombie", which carries out the actual attack. A machine installed with a "Master" program is able to communicate with a number of "Zombies" and serves as a control-handler of the attacker. An attacker can command several "Masters" directly, and "Zombies" are activated by these "Masters" at the designated time for an attack. Fig (2.5) shows this three-layer control. The reason for using such an architecture is to keep the attacker safe and difficult to trace. Now, all the preparation has been accomplished. The attacker only needs to cross his fingers and wait for an appropriate time to launch his DDoS attack. When a defending server suspects that it is under a DoS attack, it can only find numerous legitimate connection requests received from a large number of legitimate IP addresses, consuming all the resources of the server. However, the real owners of these "Zombies" are unwitting accomplices [34], and do not know what has actually happened on their machines.

The improvements in DDoS attacks can be summarised as having two main features.

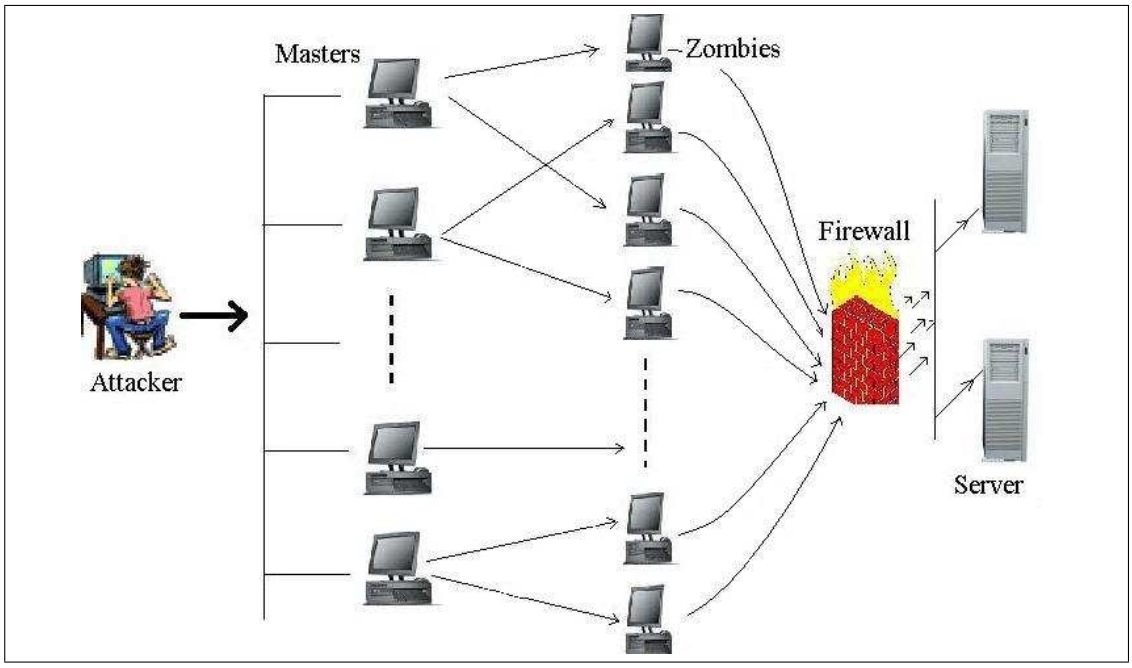

Figure 2.5: Three-layer Control For A DDoS Attack

One is that DDoS attacks may effectively bypass IP spoofing defence mechanisms. Before 2000, researchers exploited ingress/egress filtering edge routers to stop most packets with spoofed IP addresses. This forced the attacker to perform IP spoofing by using only the addresses from his own network. A simple and powerful solution against these attacks is to filter all packets from one suspected network in order to ensure service for legitimate clients from the rest of the Internet. Unfortunately, DDoS attacks can pass around this filter by launching attacks from different networks. It would be pointless for a network service provider to block all request packets from so many networks.

The other characteristic is that a DDoS attacker can amplify attack traffic immensely by using self-propagation worms to compromise sufficient computers on the Internet. He can manually or automatically scan the Internet to find each vulnerable machine on N networks as his "Zombie". One "Zombie" issues $1/N$ traffic load to a target server. If N is big enough, it may bring down any target and cause an incredible amount of damage.

DDoS Tools and Technologies

Usually, attackers utilise professional tools to launch attacks. Sub7, TFN, Trin00 and "Stacheldraht" are earlier DDoS tools, whilst Kaiten, GTbot, sdbot appear to be more recent [34]. Nearly all DDoS tools, however, need some core techniques to accomplish attacks, such as scanning, propagation, and communication. These techniques are usually exploited before real attacks start, but serve as vital points in DDoS attacks. In the following section, we will describe these methods. It is beneficial for readers to comprehend how attackers recruit and control their "Masters" and "Zombies":

(1) **Scanning**

Scanning is the first step in launching DDoS attacks. What attackers scan for are vulnerable machines/systems existing within the entire Internet. "Vulnerable", in this context, means that these machines/systems are subject to intrusion attacks, since most of them have weak or even no defence systems, such as firewall or anti-virus software. Some of them may have a number of system bugs (for example, bugs reported in Microsoft Windows systems or Internet Explorer), and have not been fixed in time. All of them offer the opportunity for attacks to intrude these machines/systems and leave unnoticeable codes for malicious intentions. In early DDoS attacks, an attacker had to personally scan and identify all the potential targets he/she required, storing their addresses into a list. This list was used to direct compromised machines to recruit more "Masters" and "Zombies". More recently, this situation has been changed by network worms like Code Red, which can fulfill the process of scanning-detection-infection-propagation automatically, without any direction from attackers [63].

A scanning strategy is a method for selecting the next machine to be probed. A primitive type is random scanning strategy, in which compromised machines probe random IP address for potential targets. Sometimes this technique can lead to network congestion, since many machines may detect, and try to intrude, the same IP address simultaneously. Hitlist scanning can avoid this problem by recording all the machines that have been detected and compromised. This technique is utilised to speed up the initial slow phase of worm propagation. More details and other scanning strategies can be found in [39, 63].

(2) **Propagation**

Today, automated propagation has been developed into three general models [14]:

the central source propagation, the back-chaining model and the autonomous model. In central source propagation, the attack code is stored in a central server or set of servers. A propagation is fulfilled by an intrusion transferring a copy of the attack code from the central source to a newly compromised system. During back-chaining propagation, a system which is intended to compromise other potential targets, serves as a central source from which the attack code can be delivered to others. The newly infected machines then become the source for the next propagation [39]. In contrast with the central source propagation, this model ensures a smooth delivery for the attack code. Autonomous propagation can directly inject a vulnerable machine with the attack code, without downloading or copying it from a external source, saving the file retrieval phase.

(3) **Communication**

Communication mechanisms are another important issue not only for DDoS attackers, but also for security engineers. If communication packets from an attacker to "Masters", or from "Masters" to "Zombies", can be detected ahead of real attacks, according to the destinations of these packets, the compromised machines can be identified and removed easily. However, as time goes by, communication channels for DDoS attacks are becoming more difficult to detect. Early DDoS tools used TCP/UDP packets for communication, which are relatively easy to identify using network monitoring tools, such as Intrusion Detection Systems (IDS). Then attackers found the Internet Relay Chat (IRC) provides a sufficiently anonymous environment for communicating with "Zombies" directly, which makes it more difficult to identify DDoS networks [14].

Attack Network Topologies

There are two major topologies in DDoS attacks: direct attacks and reflection attacks.

The architecture of direct attacks has been demonstrated in Fig (2.5). An attacker controls several "Masters" that are responsible for transmitting the attack command to a number of "Zombies". At a designated time, all the "Zombies" launch a direct DDoS attack by flooding a victim with numerous bogus requests. The attack command flows along the following path:

Attacker → **Masters** → **Zombies** → **Victim**

The architecture of a reflection DDoS attack is illustrated in Fig (2.6). Denote network (**a**), which consists of all the "Masters" and "Zombies", as the attack network. Network (**b**) represents a large number of well-meaning and innocent servers, which unfortunately act as reflectors in a reflection DDoS attack. Many of these reflectors possess broadband or good connectivity, such as Internet server providers, TCP servers, and so on. In this kind of attack, by manipulating attack network (**a**), an attacker can mount many initial requests carrying a target's IP address to innocent systems in network (**b**). These systems will unwittingly reply and return corresponding messages to the victim, which can easily exhaust the victim's bandwidth and lead to severe network congestion. In reflection DDoS attacks, it is more difficult to detect malicious packets on the Internet, and much harder to find clues to the attack network or the attackers' real IP addresses.

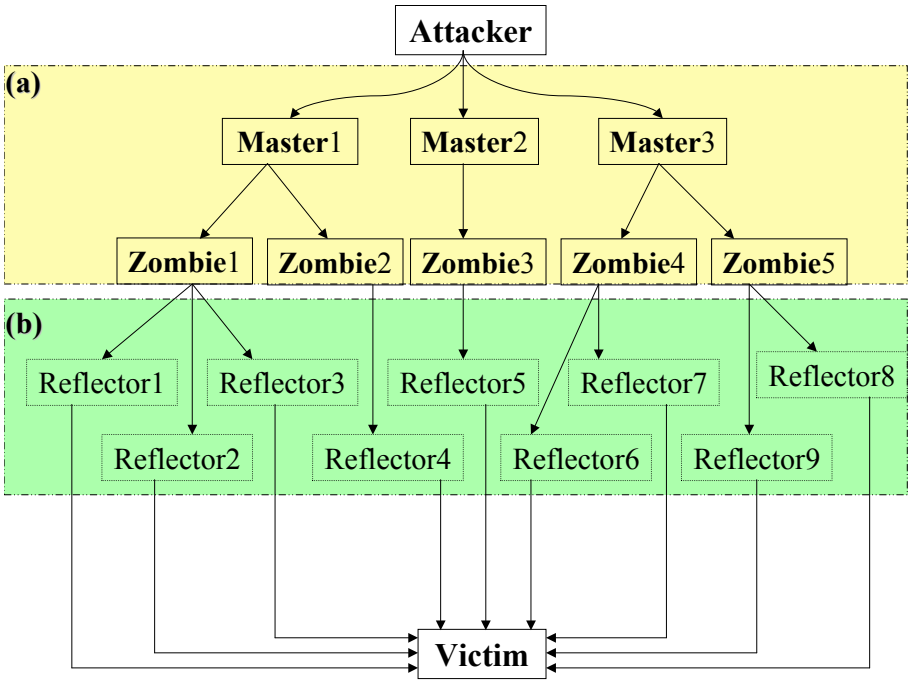

Figure 2.6: Reflection DDoS Attack

2.5 Proposed Countermeasures

To prevent DoS attacks, many defence mechanisms have been proposed. Various firewalls and router configurations have been suggested by network administrators and commercial vendors. Protocol designers are also trying to improve existing protocols to make them resistant to DoS attacks. However, most of them inevitably have potential disadvantages, and are not capable of successfully defending against DoS/DDoS attacks.

Before introducing several proposed countermeasures, we look briefly at the differences between the Internet and the traditional phone network[2], which does not often suffer from malicious denial of service attacks inside [15]. There are three reasons for this. First, every connection request (call) binds tightly with its real address (phone number). It is fairly easy to identify an attacker. Second, it seems impossible in the phone network where an attacker exploits worms/viruses to compromise other telephones for a cooperated DoS attack. The last reason is that a DoS attack launched via the traditional phone network requires a lot of resources, including human resources, money and time.

According to these reasons, scientists and researchers strive to find similar ways to resolve DoS problems. Ingress/Egress filtering and packet marking can be used to obtain a relatively reliable IP address. Client puzzles are deployed to increase the cost of launching DoS attacks. A number of anti-virus softwares and intrusion detection systems have been developed to detect and stop the spread of network worms.

In this section, we provide a brief introduction to current defence methods. More information can be obtained from the following literature: Trackback IP [44, 42], Ingress /egress filtering [25, 43], SYN cookies [8, 11], Client Puzzle [5, 6, 15, 33], and Operating system improvement [13, 49].

2.5.1 Ingress/Egress Filtering

The aim of filtering is to stop packets with spoofed IP addresses from reaching a target server. To date, two primary methods have been studied: ingress filtering and egress filtering [67].

[2]Internet phone services may attract DoS attacks toward the traditional phone network, by exploiting software (such as Skype, Net2Phone, etc) to generate enough call requests to block a call centre. However, in this thesis, we treat Internet telephony as an extended application of the Internet and as such is vulnerable. As a result, we believe that if DoS attacks can be solved for the Internet, the possibility of the traditional phone network being attacked by Internet telephony will decrease.

Ingress filtering is applied on the external interface of a network (e.g. firewall/routers) and drops all suspected incoming packets. For example, if the source address of an incoming packet belongs to its internal network, this packet will be dropped immediately. This scenario is illustrated in Fig (2.7), where three packets from distinct IP addresses try to pass into a subnet, and the firewall filters unwanted packets according to the ingress filtering rule.

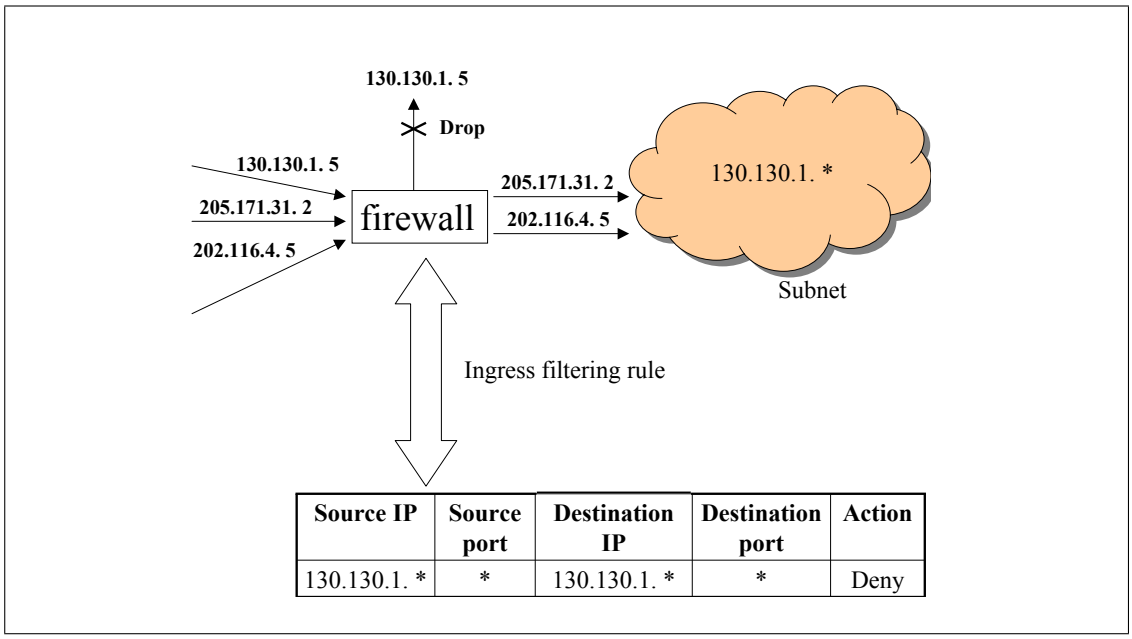

Figure 2.7: Ingress Filtering

In contrast, egress filtering is exploited on the internal interface of a network to inspect packets going out. It will filter the packets that do not have the local network addresses.

- **Advantage:** The packets filtering mechanism is an efficient way to prevent most spoofed packets from travelling on the Internet. Although an attacker can still perform IP spoofing by using his/her local network addresses, it is relatively easy for victim servers to trace back and identify the deployed network, then tackle it with corresponding security measures.

- **Disadvantage:** As mentioned in Section 2.4, IP spoofing may help attackers hide their real identities, which, as a result, becomes an important step in launching a DoS/DDoS attack. However, DDoS attacks, in particular, are often launched from real IP addresses ("Zombies" and "Master"). Ingress/Egress filtering do

not work well in these DDoS attack scenarios. Moreover, to efficiently prevent spoofed packets, filtering mechanisms must be applied widely on network routers and firewalls within the whole Internet, which would be not easy to reach in today's Internet, where fills in numerous different network devices based on various network protocols and industry standards.

2.5.2 Packet Marking

Packet marking is also used to prevent IP spoofing. In this method, a packet can be traced back to its source address by inserting traceback data into the packet when it passes through distinct routers to the destination [42, 44]. If a victim is attacked, it can deduce the path of malicious packets in order to identify the attacker's source address.

- **Advantage:** This may be combined with filtering mechanisms to destroy IP spoofing completely. Packet marking is capable of meeting the demands of DDoS defence.
- **Disadvantage:** Requiring each router to insert unique information as a packet passes will increase traffic load and create much information redundancy. Moreover, since the traceback data needs to be encoded, packet marking encounters computational difficulties when it has to deal with a large number of attack packets [67].

2.5.3 Time-out

In the time-out approach, a server deploys a short period of time to wait for the final ACK message, which should be returned from a client to complete TCP connection protocol. After this time, which we call "time-out", the connection request will be rejected. Meanwhile, the distributed buffer space for this connection will also be cleared [33].

- **Advantage:** This approach may help the server to prevent too many half-open SYN packets from being crammed into the buffer space. It is easy to implement in existing servers, without any need to increase software/hardware, or modify network protocols.
- **Disadvantage:** This approach can be overwhelmed by a SYN flooding attack with a high-speed rate, which means the buffer space may be filled with malicious

connection requests before each time-out occurs. Furthermore, a short time-out can possibly influence the service availability of clients whose network connection has a long time delay.

2.5.4 Random Dropping

In the random dropping approach, a server selects a certain percentage for its buffer capacity, which should never be 100 percent. When the amount of consumed buffer space reaches this designated value, a number of half-open connection requests in the waiting queue will be rejected at random.

- **Advantage:** Using this approach, a server is able to avoid a complete denial of service, since the server buffer will never be consumed completely. The server only needs a random dropping algorithm to perform this approach.
- **Disadvantage:** No identification for random dropping may result in a substantially degraded service for legitimate clients. It is a undesirable consequence, especially when most requests waiting in queue belong to a DoS attacker.

2.5.5 SYN Cookies

Since TCP SYN flooding attacks exploit the inherent shortcomings of the protocol, it appears reasonable to improve the protocol to resist attacks. SYN cookies belong to this category.

In the SYN cookies approach, a server verifies the authentication of connection requests by using so-called "cookies", which are computed by hashing a series of connection parameters. These parameters include a client's IP address, port number, and a secret number known only by the server. When receiving a client request i, the server generates a secret number, and hashes connection parameters to obtain a "cookie" H_i. The server then returns a SYN-ACK message containing H_i to the client. Until the server receives the final ACK message with the correct H_i, the resource will not be allocated for this formal connection [33].

- **Advantage:** SYN cookies are believed to be one of the most successful defences against TCP SYN flooding. Instead of allocating resources without any verification, SYN cookies introduce a small authentication mechanism to help the server

distinguish between spoofed IP addresses and legitimate ones. Resources are only granted to the client who can successfully pass the authentication.

- **Disadvantage:** The major limitation of this approach is that SYN cookies assume that IP spoofing attackers are incapable of eavesdropping on the SYN-ACK messages sent to the spoofed IP address, and consequently, attackers cannot provide the server with the correct cookies. This assumption may not be always correct. For example, if spoofed IP addresses are located within the same subnet, it is relatively easy for an attacker to intercept all the packets transferred on the network.

  On the other hand, the cryptographic hashing used in SYN cookies is quite expensive. Some engineers argue therefore, that servers expecting a lot of incoming connections should not use this solution.

2.6 Summary

Since 2000, the more serious impact of DoS attacks on the Internet has caused public suspicion about the Internet as a feasible tool for electronic commerce and other electronic businesses. Unfortunately, as mentioned above, few proposed defence systems have been found capable of protecting both network servers and legitimate clients against DoS attacks. The situation seems hopeless, but it may be saved by the magic of client puzzles, which need not sacrifice any legitimate clients to the server's interests (as Random Dropping does), they do not lead to a heavy traffic load on the Internet (as Packet Marking does), and they can even be based on a stronger assumption than SYN cookies. Client puzzles are the major theme of this thesis, and they will be elaborated on from the following chapter.

Chapter 3

Client Puzzles

Client puzzles are proposed as a promising countermeasure against DoS attacks. Since these attacks mostly exploit defects in existing network protocols, the advantage of client puzzles over other proposed methods is that by improving protocols directly, it is feasible to confine DoS attacks in a harmless range for defending servers. This chapter begins with an overview of client puzzles, in which several features of client puzzles will be addressed. In addition, the idea of client puzzles deduces from cryptographic puzzles that were first used in key agreement [38]. To assist readers with a comprehensive knowledge of cryptographic puzzles, their two applications in relevant security fields will be introduced. Then, the *Client Puzzle Protocol* proposed by Juels and Brainard [33], which is designed as a work platform for client puzzles, will be examined. As how to construct proper puzzles for DoS defence is the key issue of client puzzles, two proposed puzzle constructions will be described at the end of the chapter. Meanwhile, their weaknesses are analysed in detail.

3.1 Overview of Client Puzzles

In a typical DoS attack, an adversary deploys unauthorised service requests to consume the limited resources of a target server. The aim of client puzzles is to impose a moderate authentication cost on each client wishing to obtain service from a defending server. This can be achieved by asking clients to solve different cryptographic puzzles (here we call them *client puzzles*). "Cost", in this context, means computational cost, such as CPU processing time and memory space. Recent studies show that existing DDoS tools are designed carefully not so as to disturb "Zombie" computers, and thus avoid alerting their real owners. In other words, "Zombies" are unable to furtively

compute puzzles for the adversary. In this way, the defending server may force the adversary to give up, because applying for more resources demands that the adversary invest more resources himself/herself.

A client puzzle is a moderately difficult cryptographic problem. Unlike conventional public-key authentication, creation and verification of these puzzles is much easier and less expensive, which can satisfy the demands of dealing with large amounts of incoming requests. The cost of computing client puzzles is negligible for legitimate clients, but unendurably expensive for DoS attackers, who attempt to acquire considerable resources from the server.

Client puzzles need a work platform to fulfill their promise. In 1999, Juels and Brainard successfully proposed a client puzzle protocol to defend against TCP SYN flooding attacks. The idea of the client puzzle protocol is very simple [33]. When there is no evidence of attack, a server accepts connection requests normally. If the defending server comes under a DoS attack, it distributes a unique client puzzle to each client who is applying for a connection. In order to obtain the server's resources for his/her connection, a client must compute the puzzle correctly, and return the solution in time. This protocol is deployed in conjunction with the traditional time-out, which is used to control the time period for puzzle computation. Consequently, it is hard for an adversary to compute large numbers of puzzles in a short period, which can be used to differentiate legitimate requests from malicious half-open ones.

The client puzzle protocol has several advantages over other defences. First, it improves defects in existing network protocols, and does not increase packet load on the Internet. Second, this protocol is working within a stronger attack model than standard measures [33]. For example, SYN cookies are based on the assumption that an attacker is incapable of intercepting messages sent to spoofed IP addresses. We know that this assumption is not always correct. Third, when dealing with high-speed requests, the client puzzle protocol filters malicious ones more effectively than random dropping. The difficulty of client puzzles can be adjusted flexibly to adapt to different strengths of attacks. The last is that the client puzzle protocol can either work as an independent protocol in the application layer, or can be built into other service protocols it attempts to defend, such as TCP/IP, TLS [21], and so on.

On the other hand, client puzzles have one inadequacy. A client who requests a service from a defending server has to install a small client-side program for the computation of puzzles. Most of the other countermeasures against DoS attacks, such as Traceback IP, SYN cookies, and Ingress/Egress filtering methods, merely demand a

modification to a defending server or a fundamental network protocol, although the potential disadvantages of these approaches are inevitable, as outlined in Chapter 2. Compared with these, client puzzles are able to ensure service quality and protect servers against DoS attacks effectively, as long as clients install a puzzle-solving software. In today's network technology, such software can easily be implemented by a plug-in of a web browser, or distributed by the servers. Hence, the requirements for this special software should not be a problem.

3.2 Application of Puzzles

Client puzzles are viewed as special cryptographic problems. In fact, any cryptographic problem can be a puzzle. But not all of them can be client puzzles. Cryptographic puzzles have been applied in a wide range of security fields, such as defending against junk e-mail [22] and metering Web site usage [26]. In this section, we shall address two of these applications in security fields to help readers gain a comprehensive knowledge of puzzles.

3.2.1 Puzzles in Key Agreement

Cryptographic puzzle is an old technique, which was first exploited by Ralph C. Merkle [38] to solve the problem of secret key exchange. In the context of public key cryptography, Merkle raised the possibility of transmitting a secret key in insecure communication channels.

Informally, the method can be simply described as follows: $\mathcal{A}$ and $\mathcal{B}$ want to communicate over an insecure channel, and $\mathcal{C}$ has the ability to eavesdrop on all the information transmitted between $\mathcal{A}$ and $\mathcal{B}$. Assume that $\mathcal{C}$ can neither forge messages, nor interrupt the communication. In addition, $\mathcal{C}$ is hard to predict the new information generated by either $\mathcal{A}$ or $\mathcal{B}$.

$\mathcal{A}$ generates a puzzle set which contains n cryptographic puzzles. This set is then sent to $\mathcal{B}$ who randomly picks up one puzzle from the set, and solves it. Both $\mathcal{A}$ and $\mathcal{B}$ now possess a common piece of information – the solution to the puzzle that $\mathcal{B}$ solved. They will refer to this solution as the secret key for their further privacy. We assume that one unit of computational cost generates each puzzle, and n units of computational cost solve each puzzle. This implies that both $\mathcal{A}$ and $\mathcal{B}$ have invested n units of effort. However, the eavesdropper $\mathcal{C}$ is unable to find out which puzzle $\mathcal{B}$

selected. In consequence, $\mathcal{C}$ must consume n units of effort to solve each of n puzzles, for a total of n^2 units of effort.

According to Merkle's method, an adversary is forced to expend an amount of computational overhead which increases as the square of the effort needed by the two communicants for selecting a key.

3.2.2 Puzzles in Junk Mail Defence

Due to the ease and low cost of sending electronic mails, and particularly the simplicity of broadcasting the same message to a group, junk mail has becomes a practical threat on the Internet. People feel frustrated and helpless when they encounter the same situation every day: hundreds of meaningless mails in their e-mail boxes. In a worst-case scenario, an adversary can easily render an arbitrary e-mail address incapable of accepting other mails, by flooding numerous junk mails to exhaust its capacity.

In 1992, Dwork and Naor introduced cryptographic puzzles into junk mail defence [22], in which the mail system requires a sender to pay a moderate cost for each message. The cost can be defined as a hardware/software resource consumed for computing a cryptographic puzzle, and it is trivial for legitimate clients, but expensive for a junk mail sender. This was the first time that cryptographic puzzles were incorporated into access control.

Besides proposing a framework of defence systems, the most valuable contribution of their paper is to describe the definition and properties of a "puzzle" f [22]:

1. f is moderately easy to compute for clients.
2. f is not amenable to amortisation: given l values $m_1, ..., m_l$, the amortised cost of computing $f(m_1), ..., f(m_l)$ is comparable to computing $f(m_i)$ for any $1 \leq l \leq l$.
3. given x and y, it is easy to determine if $y = f(x)$ for mail servers.

Note that the second point provides a definition for the hardness of computing a puzzle which can be used to distinguish degrees of computation difficulty. According to this definition, a puzzle may be chosen to act as a trapdoor algorithm [51]: given some additional information (the trapdoor), the computation would be considerably less expensive. It is convenient for puzzle distributors to verify the answers later.

Dwork and Naor also presented several puzzle constructions to support their system. Among these, the hash function is extended respectively by Juels [33] and Aura [5] in

their papers.

3.3 Client Puzzle Protocol

To defeat TCP SYN flooding attacks, Juels and Brainard [33] proposed a new scheme to remedy the flaws in the TCP connection establishment protocol. This is the client puzzle protocol, which is referred to as a fundamental infrastructure of client puzzles. Nearly all relevant research on client puzzles (including ours) is based on it. This section will introduce the specifications of this protocol, along with its work environment.

Juels and Brainard designed an attack model in which the client puzzle protocol might work effectively. They first assumed that an attacker $\mathcal{A}$ attempts to make use of a client/server protocol M so as to deplete either the memory or computational resources of a target server S. Meanwhile, a number of legitimate clients $\{C_i\}$ exist in the same network, and may ask for service from S. They then addressed several assumptions to define the capability and incapability of $\mathcal{A}$. We explain and discuss three of the major assumptions in the following section.

(1) *$\mathcal{A}$ cannot modify packets sent from any C_i to S.*

If this assumption does not hold, then any adversaries who are able to modify packets arbitrarily can launch a DoS attack simply by changing the source addresses of all packets sent to the server. By doing this, no legitimate client can obtain a response from the server, because all messages that have been replied will be sent to false addresses. Hence, the attackers do not need to mount resource depletion attacks, and can easily defeat the server. On the other hand, it is possible that a limited number of clients' packets may be modified by adversaries on the Internet. But, so long as it is not on a large scale, this protocol can still protect the servers against DoS attacks.

(2) *$\mathcal{A}$ cannot significantly delay packets sent from any C_i to S.*

Similarly, if any attackers can delay packets transferred on the Internet for a long while, it is equivalent to interrupting the communication line. Attackers do not need to launch any resource depletion attacks, and can easily achieve the goal of denial of service.

(3) *$\mathcal{A}$ can read any messages sent to any IP address.*

In fact, we do not completely agree with this assumption. Since a packet may be sent to a large number of different networks, it is hard for attackers to eavesdrop on each of them. This assumption, in fact, makes the SYN cookies method totally pointless. Currently, however it is still a practical and easy security measure against DoS attacks for many websites. Hence, in our solution, we consider that some of packets can be eavesdropped, rather than all.

Juels and Brainard tried to use the client puzzle protocol to defend an arbitrary protocol M (such as TCP or SSL) against DoS attacks. As mentioned above, this new protocol can be layered either independently on top of the protocol M, or can be embedded directly into M. In their paper, they chose the former.

The client puzzle protocol can be illustrated in Fig (3.1) and Fig (3.2). To simplify the description, we assume that the protocol M is initiated by clients, such that the first message in the protocol M is sent from the client C_i to the server S. Hence, C_i first sends a service request message to S, along with a query about whether S is under an attack. If there is no such a threat, S replies with the answer "No", and allocates memory space to continue the remaining part of the protocol M as indicated in Fig (3.1).

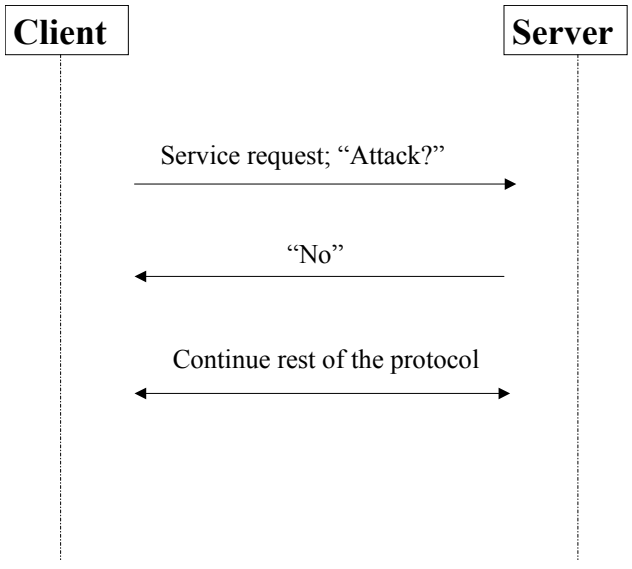

Figure 3.1: No DoS Attack Threat

If its memory usage is beyond a designated percentage, the server S may suspect that it is suffering a DoS attack. In this case, the message return to the query includes

the answer "Yes", a client puzzle and a timestamp t. C_i must resolve the puzzle correctly, and return its solution within a time period $T_d = T - t$, in which T is the current time, and T_d is designed by the server administrator. When S receives the solution, it first checks whether the solution has expired, then verifies its correctness. Only the request which belongs to the client who correctly solves the puzzle can be granted to the server's resources and continue the protocol M. In addition, the time for using the allocated memory for each connection is limited by T_c. After this time, the connection is disrupted by S, and C_i has to make another request for a new connection.

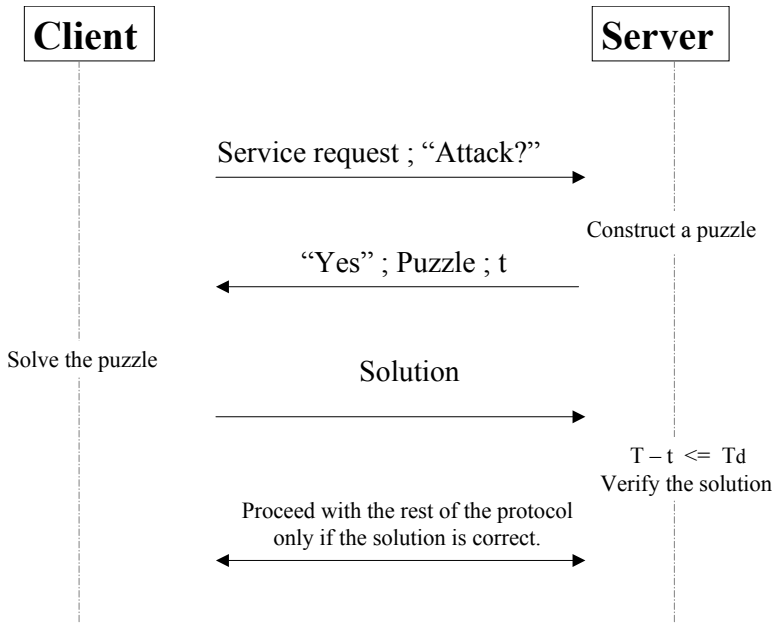

Figure 3.2: Under A DoS Attack

Juels and Brainard also presented a simple design for the puzzle, which is based mainly on a non-invertible hash function. However, in their paper, the puzzle scheme they have proposed has some unresolved problems (such as redundancy and solution collision), and needs to be improved further.

3.4 Puzzle Construction

Although the depth of research has been enhanced, and client puzzles are showing promise against DoS attacks, they still face many challenges. For instance, one of the

key issues is how to build up puzzles that are qualified to satisfy the requirements for handling large amounts of suspicious service requests. To date, researchers have proposed several puzzle construction schemes, such as the hash function based puzzles [15, 21, 33], the Diffie-Hellman based puzzles [6], the game theory based client puzzles [7], and so on. Most of them, however have underlying weaknesses that may be captured by attackers and lead to DoS attacks. In this section, we shall describe two proposed client puzzle schemes, and try to analyse their inherent problems. Following these discussions, we shall present our solutions and new schemes in later chapters.

3.4.1 Hash Function-based Puzzle Scheme

Compared with the scheme proposed by Juels and Brainard, Aura and Nikander later presented a more convincing specification of one-way hash function for client puzzles [5], in which a defending server sends a puzzle's parameters to a client, and according to these parameters, the client performs a brute-force search to find the correct solution. The difficulty of this puzzle can be adjusted by varying the puzzle's parameters. The server commits its resources to the client only after the solution is verified.

The hash function can be demonstrated as shown in the following table [5].

$$h(C, N_S, N_C, X) = \underbrace{000...000}_{\text{the } \boldsymbol{k} \text{ first bits of the hash}} \; Y$$

h = a cryptographic hash function (e.g. MD5 or SHA)

C = the client identity

N_S = the server's random number

N_C = the client's random number

X = the solution of the puzzle

$\boldsymbol{k}$ = the puzzle difficulty level

$000...000$ = the K first bits of the hash value ; must be zero

Y = the rest of the hash value ; may be anything

Figure 3.3: Aura's Client Puzzle Construction

To construct new puzzles, the server generates a random number N_S periodically, and sends it to its clients. The valid time for N_S should be short (such as 60 seconds)

in order to prevent attackers from precomputing solutions. The server also decides the difficulty level k of the puzzle. Now, N_S and k together form the puzzle that is sent to clients.

To solve the puzzle, the client also needs a random number N_C, the main purpose of which is to reuse the server N_S for new puzzles. According to the parameters received from the server and N_C and C, the client performs a brute-force search to find the solution X, which should ensure the first k bits of the hash value are all zero.

The **advantages** of this puzzle scheme are enumerated as follows:

(1) Before the server verifies the solutions, no memory space needs to be distributed to clients. The server only generates one random number N_S to create new puzzles.

(2) The only efficient way to solve the puzzle is to try every possible X until a solution is found. The cost of solving a puzzle relies on the value of k, which can easily be controlled by the server.

(3) The solutions are hard to precompute because of the short valid time for one single N_S.

On the other hand, we have also found some **shortcomings** in this scheme:

(a) The server has to keep a record of the used N_S in order to prevent attackers from reusing them and precomputing solutions. Furthermore, due to the randomness of N_S, the server has to check that a certain N_S has not been calculated before, which wastes processing time.

(b) Although the difficulty level k can easily be adjusted by the server, the computation cost for clients is relatively hard to predict, and the waiting time for the solution is similarly unknown.

(c) The vital weakness of this scheme is located at the verification phase. To verify the solution of a puzzle, the server first has to check that N_S is valid, and that C, N_C, and N_S have not been used together before. After that, the server performs a similar cryptographic hash computation as the client did, to check whether the solution is correct. This opens up the opportunity for the puzzle verification to become a target of DoS attacks, in which an adversary overwhelms the server with numerous random solutions that the server has to process.

3.4.2 Diffie-Hellman Based Puzzle Scheme

In 2004, Waters and Juels [6] proposed a fresh puzzle scheme, which is based on the Diffie-Hellman key agreement, and permits the outsourcing of puzzles. To eliminate puzzle construction as a target of DoS attacks, an external service (which they call a "bastion") is introduced in their paper to help multiple servers distribute puzzles. Interestingly, many servers can rely on the same bastion, while the bastion need not know which servers are deploying its service.

Puzzle distribution using this approach depends, in particular, on virtual channels, rather than per-request or per-session as in previous schemes. A web server may limit the maximum number of open TCP connections per channel in order to control accepted communication via a restricted collection of channels, when it encounters a DoS attack. At every time interval, the bastion generates client puzzles for distinct channels, which are suitable for all servers that rely on the bastion's service. Moreover, each server has a unique identity, such as a public key. If a client intends to communicate with a certain server, he/she needs to combine the corresponding public key with the puzzle that he/she receives from the bastion, solving this reconstructed puzzle to obtain a valid token for later connection. In other words, the puzzles distributed from a bastion are general ones. Only in connection with the public key of a specific server, can the client calculate a proper solution to meet the server's demand.

According to Fig (3.4), the algorithm of the Diffie-Hellman based puzzle construction can be described as follows.

- To create a general puzzle for a certain channel c and a certain time period t, the bastion selects a random integer $r_{c,t} \in_R Z_q$ where q is a large prime. Then, select a second random integer $a_{c,t} \in_R [r_{c,t}\ , (r_{c,t} + l) \bmod q]$, in which l is the difficulty level of the puzzle. Let f' denote a one-way permutation on Z_q, and g be a published generator (order q), such that $g_{c,t} = g^{f'(a_{c,t})}$. Hence, the bastion publishes the puzzle $P_{c,t} = \{g_{c,t},\ r_{c,t}\}$.

- Receiving the puzzle $P_{c,t}$, a client (or an attacker) has to perform a brute-force testing for all the possible values in the search range, in order to find the exact a' which can satisfy the equation: $g_{c,t} = g^{f'(a')}$. Then, for a specific server's public key Y_{ID}, the solution to the puzzle is $S_{ID} = {Y_{ID}}^{f'(a_{c,t})}$.

- Using its secret key X_{ID} as a shortcut, a defending server is able to compute the solution in one step: $S_{ID} = {g_{c,t}}^{X_{ID}}$, which can be verified in the following

equations. Since

$$Y_{ID} = g^{X_{ID}} \pmod{q}$$

then,

$$\begin{aligned} S_{ID} &= Y_{ID}^{\ f'(a_{c,t})} \\ &= g^{\ X_{ID} \cdot f'(a_{c,t})} \\ &= g^{\ f'(a_{c,t}) \cdot X_{ID}} \\ &= g_{c,t}^{\ X_{ID}} \pmod{q} \end{aligned}$$

By comparing these two solutions, the server then determines whether to allocate resources to the client.

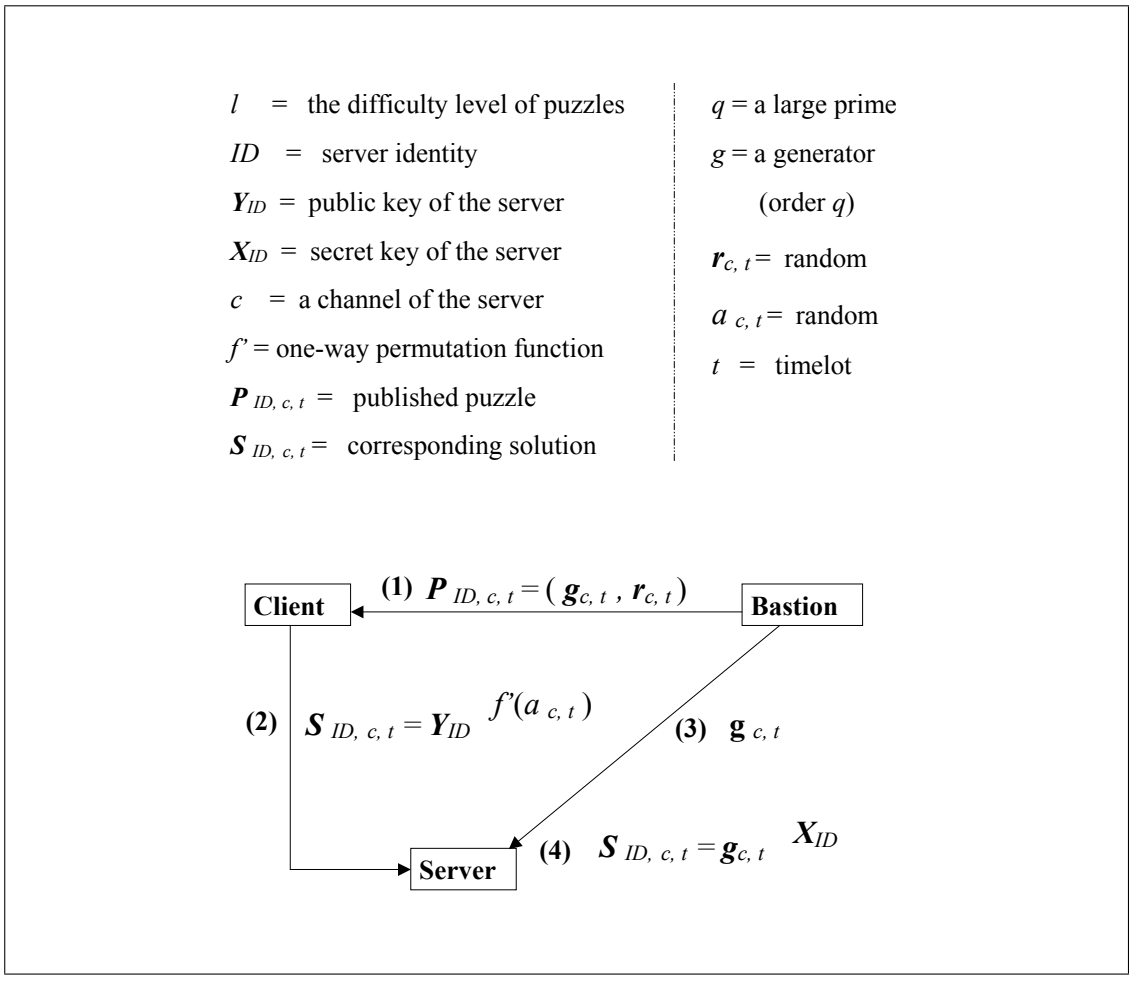

Figure 3.4: D-H-based Client Puzzle Construction

The main **advantages** of this puzzle scheme are enumerated as follows:

(1) This method allows clients to solve puzzles offline. First, a client obtains a general puzzle from a bastion, and solves it within the valid time. The client then combines this solution with a server's public key to compute the specific solution to meet the server's demand. All these computations can be achieved offline, so that clients do not have to sit and wait while their computers solve puzzles.

(2) The deployment of a bastion can mitigate the possibility of puzzle distribution as a target point of DoS attacks.

(3) The deployment of virtual channels help the server detect DoS attacks and allocate system resources rationally. By inspecting the usage of channels, the server can determine whether it is under attack, and limit communication to a restricted collection of channels.

Similarly, we have identified some **problems** in this scheme:

(a) We note that this scheme relies on virtual channels, where one channel owns only one puzzle. Consequently, in a specified time period, there are only n puzzles for the server (n is the number of valid channels). If an attacker can solve several of them, he can easily exhaust the resources of the corresponding channels. Due to the fact that there is no other identity for these solutions except the source IP addresses, the server is obliged to stop all communication in these decayed channels, and legitimate clients who solve the puzzles for these channels cannot obtain a response.

(b) Furthermore, this scheme is vulnerable to attackers' eavesdropping. If an attacker is able to eavesdrop on packets sent from clients to the server, he/she may exploit the solution computed by legitimate clients to deceive the server.

(c) Another practical problem is that all the servers that depend on the same bastion have to have uniform configurations, such as the same number of channels, and the same size of the timeslot. It is inflexible for web servers that possess distinct resource and bandwidth capability.

(d) Finally, the cost of computing a puzzle solution on the server requires one modular exponentiation, which is too expensive to be accepted in practice.

3.5 Summary

This chapter has shown the robustness of client puzzles in DoS defence. Although it has many advantages over other proposed measures, the puzzle mechanism still faces many challenges. The most important issue is what an efficient puzzle construction might look like. Many researchers have proposed puzzle schemes, but few of them can satisfy the demands of both web servers and legitimate clients. In the following chapters, we will propose one general model and two specific schemes to solve these problems. The ideas are based mostly on conventional trapdoor algorithms and several hard problems in mathematics, such as Factorisation and Logarithm Discrete.

Chapter 4

Trapdoor-based Client Puzzle System

The preceding chapter introduced the idea of client puzzles, and analysed their potential weaknesses by investigating some existing schemes. In this chapter, we shall first discuss and summarise the problems that might introduce new possibilities for DoS attacks against a defending server. After that, we shall advocate the essential requirements that all promising puzzles should follow to overcome the disadvantages of client puzzles. Based on these requirements, we shall propose a new model for puzzle distribution, called a Trapdoor-based Client Puzzle System (TCPS), which is expected to mitigate the vulnerabilities of client puzzles. This chapter can be viewed as a preliminary definition of the TCPS. In the following two chapters, we shall introduce two modified families of trapdoor algorithms that can be used to fulfill puzzle construction.

4.1 Problems

Although the employment of client puzzles in DoS scenarios appears to be promising, we have noted that there is some opposition, such as the idea expressed on some websites [19, 48]. It is argued that the current techniques of client puzzles are not necessarily better than SYN cookies because, as members of the cryptographic family, most proposed puzzle schemes inevitably have two vital shortcomings.

First, the complexity of puzzle construction and verification might turn the client puzzle mechanism itself into a victim of DoS attacks. For instance, client puzzles based on Hash functions [5] require a defending server to calculate a hash function in order to verify each answer received from clients. As a result, an attack scenario may occur in which an adversary sends numerous bogus answers that the server has to process. Another more extreme example is where a Diffie-Hellman based construction [6] requires

the server to perform a modular exponentiation for the solution of a puzzle, which consumes more resources than hashing does. It is clear that the cost of verification (for a defending server), being greater than that of generating random answers (for an adversary), may result in DoS attacks. On the other hand, a complicated construction also attracts attackers' attention. A scenario of this type of attack would be where, by utilising spoofed IP addresses, an adversary floods a defending server with false connection requests to cheat the server exhausting the resources for the construction of puzzles.

The second noticeable weakness in current schemes is that, to ensure the randomness and non-iteration of client puzzles, the defending server has to store all the correctly solved instances (for example, N_S and N_C) so that the solutions cannot be reused by attackers [5, 33]. However, as time goes by, the memory space becomes congested. Meanwhile, to guarantee the validity of puzzles and solutions, the defending server has to perform two comparisons respectively in puzzle construction and verification to examine whether the puzzles and solutions have been used before. This is undesirable in a client puzzle mechanism.

4.2 Essential Requirements

To serve as a promising countermeasure against DoS attacks, client puzzles must always satisfy the following essential requirements.

1. *The computational cost of solving a puzzle on the client's side should be always higher than that of puzzle generation and verification on the server's side.*

   Although legitimate clients will experience a slight delay in having their solution verified, an adversary flooding a server with concurrent requests has to invest larger amounts of computational resources. This increases the difficulties of such an attack in practice.

2. *The puzzles are stateless.*

   This means that a defending server must create puzzles randomly, and cannot reveal any information regarding their generation, neither within the puzzle descriptions nor their solutions. It means that knowing a number of the solutions does not help a new client solve the puzzles. In consequence, puzzles are unpredictable, and we cannot compute solutions in advance.

3. *The puzzles are easy to construct.*

   Generating a puzzle should not be expensive for a defending server. Complying with this requirement prevents the puzzle construction itself from being a victim of DoS attacks, where an adversary may flood bogus requests to exhaust the computational resources of a server.

4. *The solutions are easy to verify.*

   The reason for this requirement is similar to the one above. To protect a defending server from depleting its resources to verify a large number of random answers, the verification process should be as simple as possible. For example, the comparison of two numerical values could be one of the simplest methods, which in particular depends on the uniqueness of puzzle solutions.

5. *The complexity of puzzles is adjustable according to the strength of an attack.*

   As a puzzle's complexity increase, the cost of solving a puzzle should also increase. This gives some flexibility in dealing with the conflict between defending against DoS attacks efficiently and providing qualified and timely service for legitimate clients.

4.3 Definition

Some trapdoor one-way algorithms possess a range of useful properties that meet the requirements above. In this section, we first introduce relevant knowledge about trapdoor functions. Then, we define the proposed Trapdoor-based Client Puzzle System (TCPS), including its security properties. We demonstrate that this system meets all the requirements as stated above.

First of all, we provide some notations that will be used in the course of this thesis. We refer to *probabilistic polynomial time Turing machines* as efficient algorithms [29]. Let $\mathcal{F}(\cdot)$ denote a randomised algorithm, and $d \leftarrow \mathcal{F}(\cdot)$ denote the event that $\mathcal{F}$ outputs the value of d. We denote the probability of event B happening after $A_1, \ldots, A_n$ by $Pr[A_1; \ldots; A_n : B]$.

We say that a function $f(n) : \mathbb{N} \rightarrow \mathbb{N}$ is negligible, if for every polynomial $B(\cdot)$ and sufficiently big n, there exists $f(n) \leq 1/B(n)$. In fact, the notion of the negligible probability describes a negligible event $f(\cdot)$ that should rarely happen, even if we

consider any polynomial-time function $B(\cdot)$ a large number of times. We also denote that $\varepsilon = 1/B(n)$.

4.3.1 One-Way Function

Our proposed system utilises trapdoor one-way functions. Informally, a one-way function is designed to provide an algorithm which is easy to compute, yet computationally infeasible to reverse [51]. For example, given x it is easy to compute the value of $f(x)$, but infeasible to obtain x if given $f(x)$ in polynomial time.

Definition 1 *$f(\cdot)$ is a one-way function if for every probabilistic polynomial time adversary $\mathcal{A}$, we require*

$$Pr[\ x \leftarrow R\ ;\ y \leftarrow R\ :\ \mathcal{A}(f(\cdot),\ y) = x\ \ s.t.\ \ f(x) = y\ ]\ <\ \varepsilon$$

where R is a random number set.

4.3.2 Trapdoor One-Way Function

We note that a single one-way function is meaningless for cryptographers, as no one can decrypt a message that is encrypted by a one-way function. A trapdoor one-way function arises on the basis of solving such a problem, which is a special one-way function equipped with some secrets called "trapdoor". A trapdoor one-way function is uniformly easy to compute in one direction, but hard to reverse. In particular, if the trapdoor is known, it is also easy to compute x by reversing the function $f(x)$.

Definition 2 *$f(\cdot)$ is a trapdoor one-way function if for every probabilistic polynomial time adversary $\mathcal{A}$, we require*

$$Pr\left[\begin{array}{c} x \leftarrow R\ ;\ y \leftarrow R : \mathcal{A}(f(\cdot), y, d) = x \\ s.t.\ f(x) = y\ \wedge\ f^{-1}(y, d) = x \end{array}\right]\ =\ 1$$

where d is the trapdoor of the function $f(x)$.

4.3.3 Trapdoor-based Client Puzzle System

The aim of a trapdoor-based client puzzle system is to protect service availability transferred between a web server S and legitimate clients C against DoS attacks. The

system exploits a specific trapdoor one-way function, where a creator of puzzles (the server S) can efficiently compute the correct solution, while other solvers (legitimate clients C and an adversary $\mathcal{A}$) must perform a brute-force search to obtain the answers. The system consists of three efficient algorithms for generating random puzzles, solving the puzzles and verifying the solutions of these puzzles, respectively.

Definition 3 *A Trapdoor-based Client Puzzle System (TCPS) consists of a three-tuple of polynomial algorithms* $(Gen, Solve, Verify)$*, where*

- *Gen* : Puzzle generation algorithm used by a defending server S. It takes security parameter t and a difficulty level l as input, and outputs a random puzzle $\mathcal{P}$ along with a correct solution S_s to the puzzle $\mathcal{P}$. That is

$$(\mathcal{P}, S_s) \leftarrow \mathbf{Gen}(1^t, l).$$

  Indeed, a puzzle $\mathcal{P}$ comprises two elements denoted as $\mathcal{P} = (\sigma, [\alpha, \beta])$ where σ is a puzzle parameter, and $[\alpha, \beta]$ is a search range of length l. That is $l = |\beta - \alpha|$.

- *Solve* : Puzzle-solving algorithm. This is an efficient algorithm for a client C to solve the puzzles. On input $\mathcal{P} = (\sigma, [\alpha, \beta])$, it computes an answer S_c.

$$S_c \leftarrow \mathbf{Solve}(\mathcal{P})$$

- $Verify$: A deterministic algorithm performed by S. On input S_s and S_c, it outputs either one or zero.

$$\mathbf{Verify}(S_s, S_c) \in \{1, 0\}$$

Success: Define the success of a client solving a puzzle by:

$$\mathbf{Verify}(S_s, S_c) = 1, \quad if\ S_s = S_c\ holds.$$

The following working diagram (Fig 4.1) is a simple prototype for our Trapdoor-based Client Puzzle System (TCPS), in which a defending server takes charge of two positions: "Creator" and "Verifier", and the clients are responsible for "Solver".

Definition 4 Difficulty of a puzzle [6]
Let $\mathcal{P} = (\sigma, [\alpha, \beta])$ *be one part of the outcome* $Gen(\cdot)$ *where* σ *is a parameter for a client* C *solving a puzzle, and* $[\alpha, \beta]$ *is a search range for the answer along with the*

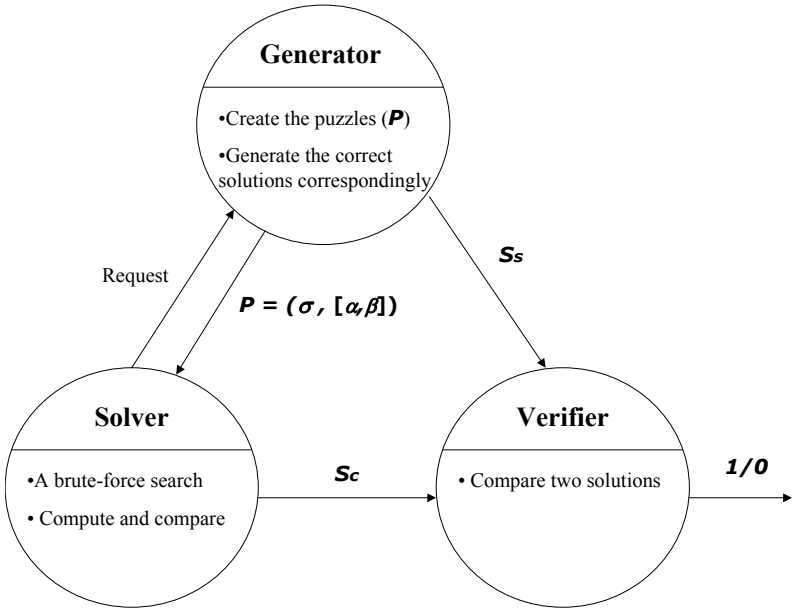

Figure 4.1: A Simple Prototype For The TCPS

width $|\beta - \alpha| = l$. *A defending server* S *sends* $\mathcal{P} = (\sigma, [\alpha, \beta])$ *to* C, *and receives the answer* S_c *correspondingly. We require that the amount of computation that a client needs to perform for the correct answer is equal to* $l/2$ *- half the length of a search range on average.*

We require that TCPS schemes should always hold the following security properties.

Correctness: We require that all the answers that are computed correctly by **Solve**$(\cdot)$, will always pass the verification. That is

$$Pr\left[\begin{array}{c}(\mathcal{P}, S_s) \leftarrow \mathbf{Gen}(1^k, l);\\ S_c \leftarrow \mathbf{Solve}(\mathcal{P}) :\\ 1 \leftarrow \mathbf{Verify}(S_s, S_c)\end{array}\right] = 1$$

Semantic Security: For every probabilistic polynomial time adversary $\mathcal{A}$, the probability of finding another efficient algorithm to compute a puzzle's solution, rather than performing a brute-force search in the given range $[\alpha, \beta]$, is negligible.

$$Pr\left[\begin{array}{c} \widehat{\mathbf{Solve}(\cdot)} \neq \mathbf{Solve}(\cdot)\ ; \\ \widehat{S}_c \leftarrow \widehat{\mathbf{Sol}(\ \mathcal{P}\ )}\ : \\ 1 \leftarrow \mathbf{Verify}(\ S_s,\ \widehat{S}_c\ ) \end{array}\right] < \varepsilon$$

Collision-resistance: The probability of the existence of two uniform integers S_c and S'_c in the search range $[\alpha, \beta]$ is negligible.

$$Pr\left[\begin{array}{c} S_c \leftarrow \mathbf{Solve}(\ \mathcal{P}\ )\ ; \\ S'_c \in [\alpha, \beta]\ ;\ S'_c = S_c\ : \\ 1 \leftarrow \mathbf{Verify}(\ S_s,\ S'_c\ ) \end{array}\right] < \varepsilon$$

Theoretically, there is only one correct answer for the puzzle in the range $[\alpha, \beta]$, which ensures that only the client who accurately complies with the description of the puzzle to perform the searching and the computation can obtain the right solution.

Information Theoretic Secrecy: A puzzle solution S_c does not reveal any information about either possible puzzle constructions or other secrets of a defending server.

$$Pr\left[\begin{array}{c} \mathcal{P}_i = (\ \sigma_i,\ [\alpha_i, \beta_i]\ )\ ; \\ S_{c_i} \leftarrow \mathbf{Solve}(\mathcal{P}_i)\ : \\ \{\sigma_{i+1}|\ \alpha_{i+1}|\ \beta_{i+1}\} \leftarrow S_{c_i} \end{array}\right] < \varepsilon$$

4.4 Security Assumption

It is well-known that the security of many existing public-key cryptosystems, such as RSA, Diffie-Hellman and DSS [35], is based on one assumption: that there is a mathematically hard problem which is difficult for cryptanalysts to solve in a polynomial time. Two prevalent hard problems widely used by cryptographers are the discrete logarithm and factorisation [64]. In the following chapters, we will employ these hard problems, combined with trapdoor functions, to develop two novel client puzzle schemes. Before that, we shall introduce definitions of these hard problems.

4.4.1 The RSA Assumption

The factorisation problem is a problem used to calculate a series of prime integers from a given number. In current cryptosystems, people particularly focus on factoring an integer N which is the product of two large primes p and q. This problem has been studied by many researchers, and even now, it still cannot be solved efficiently.

The RSA Assumption is derived from the hardness of factorisation problem, which was first introduced by Rivest and Shamir in [50]. They claimed that it is infeasible to compute e-root in the group of order $\phi(N) = (p-1)(q-1)$, where $e \in \mathbb{Z}^*_{\phi(N)}$ is relatively prime to $\phi(N)$. That is given N, e, and a random element $y \in \mathbb{Z}^*_N$, it is hard to compute x such that $x^e = y \pmod N$. More formally, we define it as follows.

Assumption 1 *The RSA Assumption holds, if for all probabilistic polynomial time adversaries $\mathcal{A}$, the following probability*

$$Pr\left[\begin{array}{c} N \leftarrow RSA(p,q)\ ; \\ e \leftarrow Z^*_{\phi(N)}\ ;\ y \leftarrow \mathbb{Z}^*_N : \\ \mathcal{A}(N,y,e) = x\ s.t.\ x^e = y \pmod N \end{array}\right] < \varepsilon$$

As a matter of fact, computing e-root in $\mathbb{Z}^*_N$ **is equivalent to** solving the factorisation problem. Assume that a polynomial adversary $\mathcal{A}$ is able to factorise N into two large primes p and q efficiently. He can then compute $\phi(N) = (p-1)(q-1)$, and exploit the extended Euclidean algorithm [9] to obtain d such that

$$e \cdot d \equiv 1 \pmod{\phi(N)}$$

That is

$$d \equiv e^{-1} \pmod{\phi(N)}.$$

Now, he can obtain $x = y^d \pmod N$ by taking the modular exponentiation instead of the e-root computation.

Hence, we believe that if the factorisation problem is hard in polynomial time, the RSA Assumption holds. However, if people can find an efficient way to solve the factorisation problem in the future, the RSA Assumption will not be valid any more.

Another confusing problem which should be noted is that the security of the RSA scheme[1] does not depend totally on the factorisation problem. This means that an

[1]The RSA scheme can provide encryption and signature functions [35], which is a different notion from the RSA Assumption.

attacker may break the implementation of the RSA without the operation of factoring N. Some of these types of attacks have been reported in [51] and [10], such as "Chosen Ciphertext Attack", "Common Modulus Attack", "Low Encryption Exponent Attack", etc.

We use Table (4.1) to illustrate the relationship between the factorisation problem, the RSA Assumption, and the RSA scheme.

the RSA Assumption	$\longleftarrow$	**Factorization Problem**	$\stackrel{?}{\longrightarrow}$	**RSA**
(hold)		(hard)		(secure)

Table 4.1: The Relationship Between Factorisation, RSA Assumption and RSA

4.4.2 The Discrete Logarithm Problem (DLP)

The Discrete Logarithm Problem (DLP) is a problem that has been researched in [36, 40]: given a large prime p, a generator α of $\mathbb{Z}_p^*$, and a random element $\beta \in \mathbb{Z}_p^*$, it is hard to find the integer x $(0 \leq x \leq p-2)$ such that $\alpha^x \equiv \beta \pmod{p}$. Many popular public-key cryptosystems are based on this assumption, such as the ElGamal system and the DSS [1, 23].

Similarly, we provide a formal presentation of the DLP, in which p is a large prime (e.g. a 1024-bit number).

Assumption 2 *We say that the hardness of the DLP holds, if for all probabilistic polynomial time adversaries $\mathcal{A}$, the following probability*

$$Pr[\ \alpha \in \mathbb{Z}_p^*\ ,\ \beta \in_R \mathbb{Z}_p^*\ :\ \mathcal{A}(p,\alpha,\beta) = x\ \ s.t.\ \ \alpha^x \equiv \beta \pmod{p}\ ]\ <\ \varepsilon$$

Chapter 5

A Scheme Based On The RSA Assumption

We begin this chapter by introducing a specific trapdoor algorithm whose security depends on the RSA Assumption. We embed this algorithm into our TCPS (Trapdoor-based Client Puzzle System) framework to implement the puzzle construction phase. We shall evaluate this algorithm before employing it to develop a RSA Assumption-based TCPS. Additionally, we shall demonstrate that this proposed scheme satisfies all the essential requirements and security conditions as stated in Chapter 4.

5.1 Algorithm

In this section, we describe an efficient trapdoor scheme whose security is based on the RSA Assumption. This scheme has been widely used earlier according to literature [17, 18, 29]. To embed it into the TCPS, we make some modifications to it in our new scheme.

Let N be the product of two large primes p and q. Maintain p and q as secrets. It is computationally infeasible to factorise N for all polynomial adversaries. Denote $\phi(N) = (p-1)(q-1)$. Let $e \in \mathbb{Z}^*_{\phi(N)}$ be a prime such that $GCD(e, \phi(N)) = 1$. g is a random integer in the group of order $\phi(N)$.

Puzzles constructed in this RSA Assumption-based scheme are derived from the following equation.

$$M = g^a \cdot r^e \pmod{N} \tag{5.1}$$

where $a \in [1, e-1]$, random $r \in \mathbb{Z}^*_N$. Given a value of M, we find that for each $a' \neq a$, there exists a unique r' such that $M = g^{a'} \cdot (r')^e \pmod{N}$ holds. We present it as

follows.

$$M = g^a \cdot r^e = g^{a'} \cdot (r')^e \pmod{N} \tag{5.2}$$

Indeed, for each a', the value of r' is equal to the e-root of $M \cdot g^{-a'} \pmod{N}$.

The trapdoor of this algorithm is the value of $x = g^{\frac{1}{e}} \pmod{N}$. In this case, we plug x in Eq (5.1) as follows.

$$\begin{aligned} M &= g^a \cdot r^e \pmod{N} \\ &= (x^e)^a \cdot r^e \pmod{N} \\ &= (x^a \cdot r)^e \pmod{N} \end{aligned} \tag{5.3}$$

Let $y = x^a \cdot r$, then we show that y equals the e-root of M.

$$y = M^{\frac{1}{e}} = x^a \cdot r \pmod{N} \tag{5.4}$$

Note that if M is kept constant, y is uniformly a constant which inherits the same property as in Eq (5.2). Given a value of y, for each $a \neq a'$, there exists a unique r' such that

$$\begin{aligned} y &= x^a \cdot r \pmod{N} \\ &= x^{a-a'} \cdot r' \pmod{N} \end{aligned} \tag{5.5}$$

Now, we can compute

$$r' = (y \cdot x^{-a}) \cdot x^{a'} \pmod{N} \tag{5.6}$$

Our scheme is such that, given a pair $(a,\ r)$, the puzzle creator obtains the values of M and y $(= M^{\frac{1}{e}} \bmod N)$. Keeping them constant, the creator varies the value of a' for each new puzzle, and meanwhile, computes r' by using Eq (5.6). Due to lack of knowledge about x, y, the solvers have to conduct three steps ruled by the creator as follows, to obtain the answer r'.

(i) Receive the values of M, $g^{a'}$, e, N and a search range for r' from the creator.

(ii) Pick up a candidate r' from the range and test whether $M = g^{a'} \cdot (r')^e \mod N$.

(iii) Repeat (ii) until all possible r' are tested or a r' that satisfies Eq (5.1) is found.

In this puzzle scheme, the computational cost consumed by the creator is quite distinct from that consumed by solvers. For the creator, it merely needs one modular multiplication $r' = \mathcal{T} \cdot a' \pmod N$ to solve a puzzle, because $y \cdot x^{-a}$ can be pre-computed and referred to as a constant $\mathcal{T}$. A solver, on the other hand, has to perform a brute-force search and a number of modular exponentiations to determine the answer. Furthermore, the creator is able to control the computational cost consumed by the solvers by adjusting the length of the search ranges.

Theorem 1 *If Eq (5.2) holds, then the solution r' computed by Eq (5.6) can be verified.*

Proof: Given the initial values of $(a,\ r)$, we obtain the following result

$$M = y^e = (x^a \cdot r)^e = (x^e)^a \cdot r^e = g^a \cdot r^e = y^e \pmod N$$

To create a new puzzle, we choose a' and obtain the solution $r' = (y \cdot x^{-a})x^{a'}$. Then we verify it in Eq (5.2)

$$\begin{aligned} M = y^e = [x^{a-a'} \cdot (r')]^e &= (x^{a-a'})^e \cdot (r')^e && \pmod N \\ &= (x^{a-a'})^e \cdot [(y \cdot x^{-a})x^{a'}]^e && \pmod N \\ &= (x^e)^{a-a'} \cdot y^e \cdot (x^{a'-a})^e && \pmod N \\ &= g^{a-a'} \cdot y^e \cdot g^{a'-a} && \pmod N \\ &= y^e \pmod N \end{aligned}$$

Theorem 2 *Under the RSA Assumption, our scheme is secure.*

Proof: Before presenting the proof, we firstly recall the RSA Assumption. N is the product of two large primes p and q. Given two elements $e \in \mathbb{Z}^*_{\phi(N)},\ g \in \mathbb{Z}^*_N$ (where e is relatively prime to $\phi(N)$), it is computationally infeasible to find a non-negative integer x such that

$$x^e = g \pmod N$$

holds, which is equivalent to factoring pq [45].

We assume there is an algorithm $\mathcal{A}$ which can output a pair of (a, r) and (a', r'), where $a \neq a'$ and $r \neq r'$, for given values of $M, g, e \in \mathbb{Z}^*_N$, such that

$$M = g^a \cdot r^e = g^{a'} \cdot (r')^e \pmod N$$

holds with a non-negligible probability.

Now, we show that an algorithm $\mathcal{B}$ intends to use $\mathcal{A}$ to break down our scheme by computing the trapdoor $x = g^{\frac{1}{e}} \pmod N$, which is assumed to be secure under the RSA Assumption. This simulation is described as follows.

After obtaining a pair of values (a, r) and $(\hat{a}, \hat{r})$, $\mathcal{B}$ transforms the following equation:

$$M = g^a \cdot r^e = g^{a'} \cdot (r')^e \pmod N$$

and obtains

$$g^{a-a'} = (\frac{r'}{r})^e \pmod N$$

Let $\theta = a - a'$. Since $a, a' < e$ and e is a prime, we note that $GCD(\theta, e) = 1$. According to the extended Euclidean algorithm [9], we can find two integers m, n such that $m\theta + ne = 1$. Now, $\mathcal{B}$ can obtain the following permutation by using the previous equation.

$$\begin{aligned} g = g^{m\theta+ne} &= g^{m\theta} \cdot g^{ne} && \pmod N \\ &= (g^{\theta})^m \cdot g^{ne} && \pmod N \\ &= [\ (\frac{r'}{r})^e\ ]^m \cdot g^{ne} && \pmod N \\ &= [\ (\frac{r'}{r})^m \cdot g^n\ ]^e && \pmod N \end{aligned}$$

If $\mathcal{B}$ is able to take e-root computation in $\mathbb{Z}_N^*$, he will obtain the trapdoor x, such that

$$x = g^{\frac{1}{e}} = (\frac{r'}{r})^m \cdot g^n \pmod N$$

Note that the probability that $\mathcal{A}$ outputs $a = a'$ and $b = b'$ is equal to $1/N^2$. Moreover, e-root computation in a cyclic group (order N) is computationally infeasible under the RSA Assumption. Hence the success probability of $\mathcal{B}$ is lower bounded by $1/N^2$. We have now completed the proof.

5.2 A RSA Assumption-based TCPS

In this section, we shall present how the above algorithm is used to develop the TCPS. We assume there are two participants involved, namely a defending server S and a client C. Additionally, S establishes a TCPS for DoS defence, and C has the ability to solve puzzles by using the parameters received from S. Now, according to the definition in Chapter 4, we divide the scheme into four phases, which are the preconstruction

phase, the construction phase, the puzzle-solving phase and the verification phase, respectively. Apart from the puzzle solving, which is performed on the client's side, these phases all run on the server.

1. **Pre-construction phase**

   An analysis of the trapdoor algorithm outlined above indicates that the puzzle creator (S) needs a number of constants to generate puzzles. Thus, to decrease computational overheads and improve efficiency in the construction phase, we introduce pre-computation into our scheme.

   The pre-construction procedure is illustrated as follows:

   (i) Determine a time interval T_d, and the size of set A denoted by n.

   There is a trade-off between T_d and n, which requires some careful thought. In fact, the server needs to find the minimum n and the maximum T_d that satisfy the following requirement: n is minimum greater than the maximum connection ability of the server in each time period. It guarantees that in the same time period no element in set A can be selected more than once. For instance, if the hourly maximum connectivity of the server is 50,000 on average, n should be greater than 50,001 with $T_d = 1$.

   (ii) Establish A.

   Variable a' for each new puzzle ($M = g^{a'} \cdot (r')^e \mod N$) is derived from A. First of all, the server creates a superincreasing sequence I [51] such that

$$I = \{\ b_i |\ 1 \leq i \leq m, m > n, b_i \in [1, e-1]\ \} \cap \{\ \forall b_i, b'_i \in I |\ \sum_{i=1}^{i'-1} b_i < b'_i\ \}$$

   This means that every item belonging to I is greater than the sum of all the previous elements. For example, {3, 7, 13, 29, 67} might be a subset of I, while {3, 5, 7, 11} is not.

   To build A, the server randomly picks up $n(< m)$ non-repeated elements from I. Then, we can obtain A such that

$$A = \{a_i |\ 1 \leq i \leq n, a_i \in I\ \} \cap \{\forall a_i, a'_i \in A, 1 \leq x \leq n |\ a'_i \neq \sum_{i=1}^{x} a_i\ \}$$

   Recalling the property of a superincreasing sequence I, we know that every instance in A cannot be equal to the sum of any arbitrary instances. Fig

(5.1) illustrates this property more clearly. Assume that {1, 3, 5, 11, 23} is a subset of superincreasing sequence I. To set up A, the server chooses four non-repeatable elements from I randomly. Then, we obtain $A = \{3, 11, 1, 5\}$. Note that we cannot obtain any element in A by summing up any other discretionary number of elements. For example, 11+1 $\neq$ 3+5, 11 $\neq$ 3+5+1, $3 + 1 \neq 5$, and so on.

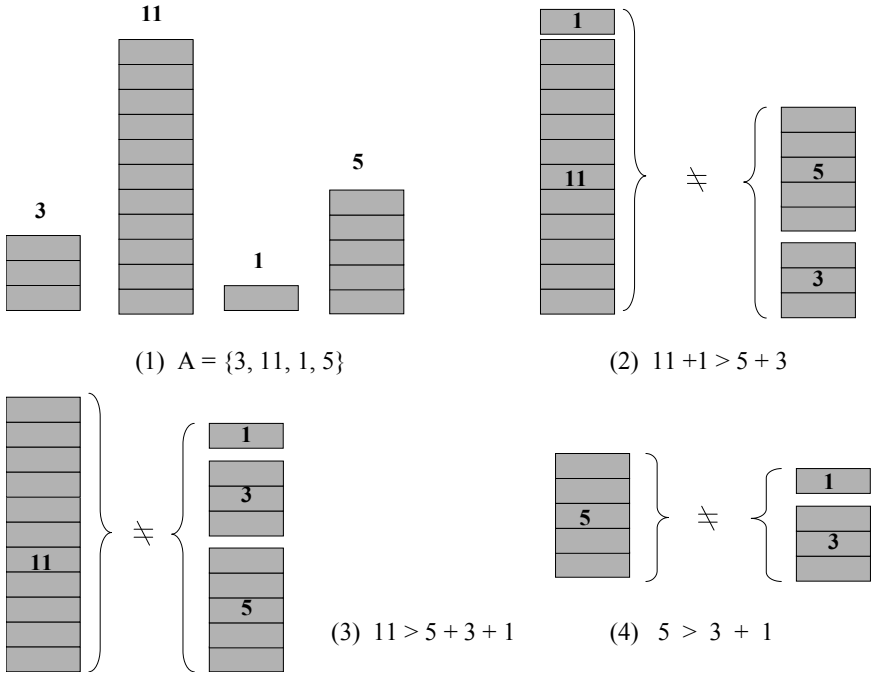

Figure 5.1: A Sample of Set A

(iii) Compute G_A, which is a multiplier factor for constructing a puzzle.

$$G_A = \{g^{a_i}(\bmod N) | 1 \leq i \leq n, a_i \in A\} \cap \{\forall a_i, a_i' \in A, a_i \neq a_i' |\ g^{a_i} \neq g^{a_i'}\}$$

(iv) Compute X_A, which is a multiplier factor for solving a puzzle.

$$X_A = \{x^{a_i} \pmod N |\ 1 \leq i \leq n, a_i \in A\}$$

(v) Generate and encrypt the time parameter t.

The form of t is defined as:

$$\{yymmddhh^1hh^2\} \cap \{hh^2 - hh^1 = T_d\}$$

For instance, when the server selects the time interval $T_d = 2$, a time parameter for 17:00 to 19:00 on January 9th, 2005 is $t = 0501091917$. Then, encrypt t by computing $g^t(\bmod N)$ and $x^t(\bmod N)$, respectively. Note that g^t and x^t are unique in an identical T_d, and computed before each new time period starts.

(vi) Calculate M and $\mathcal{T}$ by Eq (5.1) and Eq (5.6)

$$M = g^a \cdot r^e \pmod N$$

$$\mathcal{T} = y \cdot x^{-a} = M^d \cdot x^{-a} \pmod N$$

where $a \in [1, e-1]$ and $r \in_R \mathbb{Z}_N^*$ are a pair of initial values. The server will publish the value of M as part of the description of puzzles, and keep $\mathcal{T}$ secret.

After accomplishing the pre-construction, the server obtains a number of constants $\{M,\ \mathcal{T},\ A,\ G_A,\ X_A\}$, and two periodic constants $\{g^t, x^t\}$. Note that except M being public, the rest are kept secret on the server.

2. **Construction phase**

When receiving a request from a client C, the server S generates a puzzle $\mathcal{P} = (g^{a'}, [\alpha, \beta])$ along with its correct solution r_s. $\mathcal{P}$ is essential for a client solving a puzzle. r_s is stored on the server for verification of the answer returned from the client. This stage is described as follows.

(i) Compute $g^{a'}$

$$g^{a'} = g^t \cdot g^{a_i} \pmod N \tag{5.7}$$

where $g^{a_i} \in G_A$, t is the current time period and g^t is a periodic constant calculated at the beginning of t. The server picks up random $g^{a_i} \in G_A$ to generate a fresh puzzle, then marks this g^{a_i} to be unavailable until a new period comes. This means that when each period starts, all the elements in G_A are available. An element is marked unavailable throughout the same period, once it is chosen by the server to create a puzzle.

(ii) Generate r'_s

$$\begin{aligned} x^{a'} &= x^t \cdot x^{a_i} && \pmod N \\ r_s &= (y \cdot x^{-a}) \cdot x^{a'} && \pmod N \\ &= \mathcal{T} \cdot x^{a'} && \pmod N \end{aligned} \tag{5.8}$$

Note that the values of i in Eq (5.7) and Eq (5.9) are uniform.

(iii) Obtain a search range $[\alpha, \beta] \subset \mathbb{Z}_N^*$

$$\alpha = r_s - c \pmod{N}$$

$$\beta = \alpha + l \pmod{N}$$

for random $c \in [0, l)$, where l is the current difficulty level of the puzzles.

3. **Puzzle-solving phase**

Unlike the other three, this phase is performed on client's side. We assume that a client $\mathcal{C}$ has installed a specific piece of software which is distributed by the server. It solves puzzles received from the server by using a fixed equation

$$M = g^a \cdot r^e \pmod{N},$$

a triple of constants (M, e, N), and an interface for accepting puzzle parameters from the server.

When a client receives a puzzle $\mathcal{P}$, he employs an exhaustive search (brute-force) to find the answer r_c which satisfies the equation. Due to the length of the search range $l = \beta - \alpha$, a client $\mathcal{C}$ needs to perform $l/2$ modular exponentiations to find the answer r_c on average.

4. **Verification phase**

Upon receiving the answer r_c from a client, the server compares it with the stored solution r_s that has been calculated in the construction phase. If they are equal, **Verify**$(\cdot)$ outputs 1, which means that authentication of the client is verified, and the server proceeds with the rest of the request. Otherwise, **Verify**$(\cdot)$ outputs 0, and the server drops the request.

$$\textbf{Verify}(r_s, r_c) = \begin{cases} 1 & \text{if } r_s = r_c \\ 0 & \text{otherwise} \end{cases}$$

5.3 Parameter Table and Scheme Prototype

We illustrate all the main parameters used in this RSA Assumption-based TCPS in the following table, along with their corresponding properties and a simple description.

No.	Notation	Phase	Property	Public/Private	Description
1	l	1	Const*	Private	difficulty level of puzzles
2	p	1	Const.	Private	large prime
3	q	1	Const.	Private	large prime
4	N	1	Const.	Public	$N = pq$
5	$\phi(N)$	1	Const.	Private	$\phi(N) = (p-1)(q-1)$
6	e	1	Const.	Public	$GCD(e, \phi(N)) = 1$
7	d	1	Const.	Private	$e \cdot d \equiv 1 \bmod \phi(N)$
8	g	1	Const.	Private	$g \in_R \mathbb{Z}^*_{\phi(N)}$
9	M	1	Const.	Private	$M = g^a \cdot r^e \bmod N$
10	x	1	Const.	Private	trapdoor $x = g^{\frac{1}{e}} \bmod N$
11	y	1	Const.	Private	$y = M^{\frac{1}{e}} \bmod N$
12	a	1	Random	-	initial value
13	r	1	Random	-	initial value
14	$\mathcal{T}$	1	Const.	Private	$\mathcal{T} = y \cdot x^{-a} \bmod N$
15	I	1	Const.	Private	superincreasing sequence
16	A	1	Const.	Private	$a_i \in I$
17	G_A	1	Const.	Private	$g^{a_i} \bmod N$
18	X_A	1	Const.	Private	$x^{a_i} \bmod N$
19	T_d	1	Const.	Private	time interval
20	t	1	Const.	Private	$hh^2 - hh^1 = T_d$
21	g^t	1	Const*	Private	-
22	x^t	1	Const*	Private	-
23	a'	2	Var.	Private	$a' = t + a_i$
24	r_s	2	Var.	Private	Server's solution
25	c	2	Random	Private	-
26	α	2	Var.	Public	$\alpha = r_s - c$
27	β	2	Var.	Public	$\beta = \alpha + l$
28	r_c	3	Var.	Public	Client's solution

Table 5.1: Parameter Table of the RSA Assumption-based TCPS

Notation: "Const." and "Var." represent "constant" and "variable" respectively. "Const*" is denoted as a periodical constant. In particular, l varies according to the strength of an attack. g^t and x^t are changed at the beginning of a new time period. "Phase 1, 2, 3" denote "Pre-construction", "Construction", "Puzzle-solving" respectively.

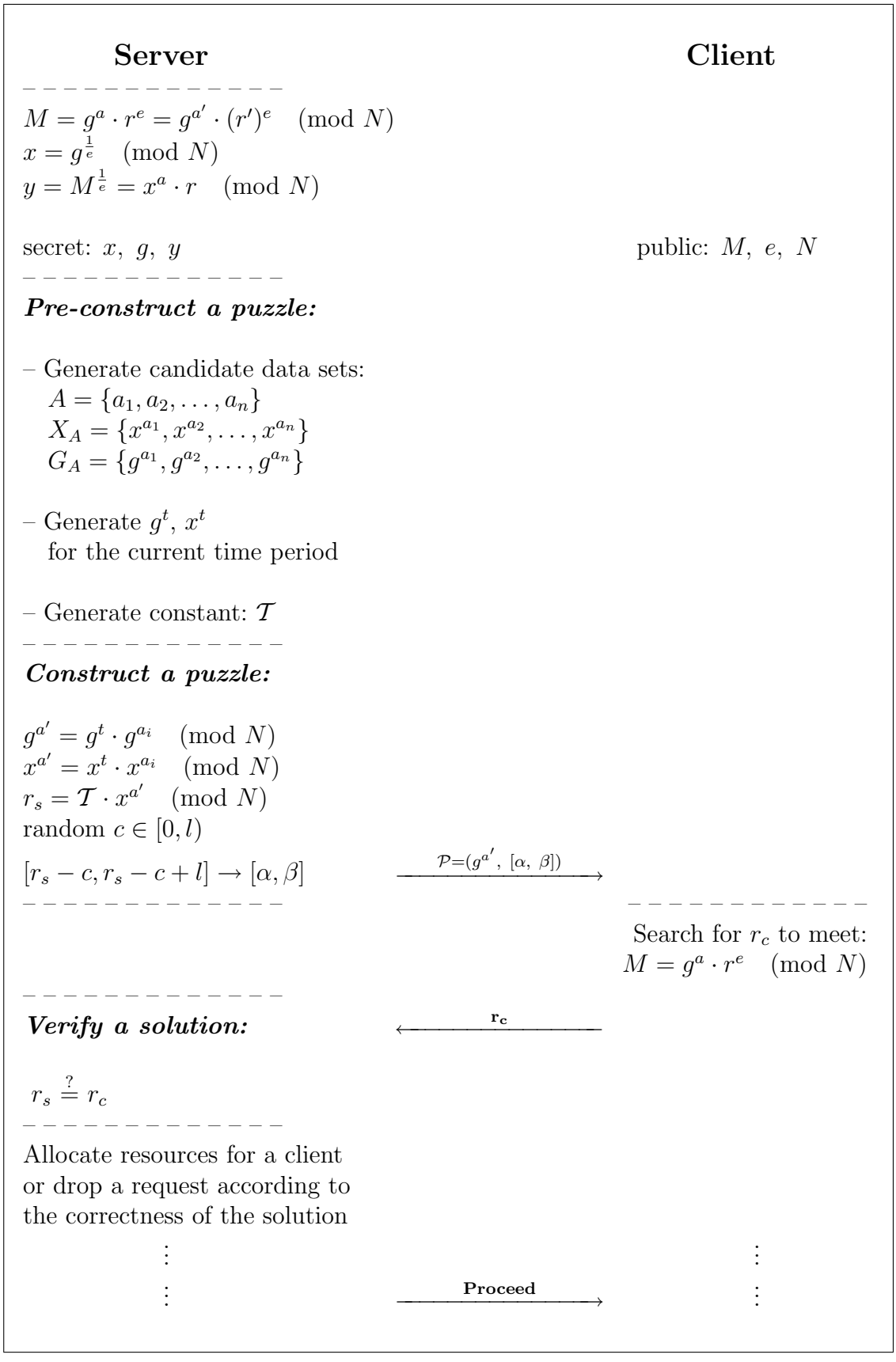

Figure 5.2: A RSA Assumption-based Puzzle Scheme

5.4 Remarks

Theorem 3 *The RSA Assumption-based puzzle scheme will produce a Trapdoor-based Client Puzzle System.*

Proof. We need to show that the above scheme satisfies the requirements of the Trapdoor-based Client Puzzle System as stated in Chapter 4.

- *Correctness.* It is easy to verify that answers computed correctly in the puzzle-solving phase can always pass the final verification on the defending server.

  In the proposed scheme, we provide a search range $[\alpha, \beta]$ which ensures the existence of the correct solution r_s. Because

  $$[\alpha, \beta] \leftarrow [r_s - c, r_s - c + l] \quad for\ c \in [0, l)$$

  where both c and l are positive integers, this means

  $$r_s - c < r_s < r_s - c + l$$

  Hence, a client can eventually find a r_c $(= r_s)$ which makes Eq (5.1) valid, as long as he employs an exhaustive search.

- *Semantic Security.* Due to the hardness of Factorisation (or the RSA Assumption), a more efficient way to obtain the correct answer than a brute-force search cannot be found. We will prove this claim in the Security Considerations section below.

- *Collision-resistance.* According to the property presented in Eq (5.2), for each $a' \in [1, e-1]$, there exists a unique value $r' \in \mathbb{Z}_N^*$ which can satisfy the equation $M = g^{a'} \cdot (r')^e \bmod N$.

- *Information Theoretic Secrecy.* No client can pre-compute a puzzle by guessing or revealing the secret components of the server.

  - If a client knows the trapdoor x, he can solve all the puzzles created by this scheme efficiently. However, we have proved in Theorem 2 that the probability of this scenario is negligible under the RSA Assumption.
  - Puzzles generated by the server can be viewed as random.

The server establishes a random and non-repeatable set A ($A = \{a_i | 1 \leq i \leq n,\ a_i \in [1, e-1]\}$) in the pre-construction phase. On the basis of A, the server calculates G_A ($G_A = \{g^{a_i} \pmod p \mid 1 \leq i \leq n,\ a_i \in A\}$) as a necessary parameter for creating puzzles. Note that the values of g^{a_i} are irregular, and hence, the probability of guessing an instance in G_A is $1/N$. When N is large enough, we can say that this probability is negligible.

This scheme also meets the essential requirements that we advocated in Chapter 4 for promising client puzzles.

Remark 1 *The computational resources consumed by a client to solve a puzzle are greater than those used by the server to generate the puzzle and verify the solution.*

A defending server performs three modular multiplications and two additions to create a puzzle and obtain the solution at the same time.

To produce $g^{a'}$, the server performs one modular multiplication:

$$g^{a'} = g^t \cdot g^{a_i} \pmod N$$

To obtain the solution, the server performs two modular multiplications:

$$r'_s = \mathcal{T} \cdot x^{a'} = \mathcal{T} \cdot x^t \cdot x^{a_i} \pmod N$$

To generate the search range $[\alpha, \beta]$, the server performs two additions:

$$\alpha = r'_s - c\ ; \quad \beta = \alpha + l$$

To verify a solution, the server only needs to make a comparison.

$$r_s \stackrel{?}{=} r_c$$

To solve a puzzle, however, a client has to conduct on average $l/2$ modular exponentiations and comparisons, which consume much more computational resources than modular multiplications and modular additions [37]. In addition, we ensure that the size of p and q are large enough to resist any attacks on factorisation. As a result, the answer can only be obtained by exhaustively searching the seed range and performing modular exponentiations until the correct instance that satisfies the public equation is found.

Remark 2 *Our scheme is better than the Hash function [5] and the Diffie-Hellman [6] based puzzle schemes.*

- The unique solution guarantees that the defending server can perform a simple comparison to verify puzzle solutions. This is more efficient and effective in protecting the verification phase against DoS attacks than the Hash function based puzzle scheme, in which the defending server is required to perform a hash function for solution verification.

- To adjust the difficulty of a puzzle, we exploit a non-negative integer l to control the length of a search range. The value of l varies according to the strength of a DoS attack. When the strength of an attack degree is low, l is correspondingly small and the cost of solving a puzzle is insignificant for a client. If an attack worsens and l enlarges, a client has to supply more resources to find a solution. On average, it requires $l/2$ modular exponentiations to solve a puzzle. Hence, our scheme is more easily measurable than the Hash function based puzzle scheme.

- To avoid puzzle iteration, the Hash function based scheme requires that a record of the used instances is kept for a long time, and it performs two comparisons in puzzle construction and verification respectively. However, relying on the time parameter t, our scheme can always obtain unique puzzles without wasting memory space and computational time. The items calculated in the pre-construction phase can also be reused in every new time interval.

- As well as relying on the outsourcing for puzzle construction, the Diffie-Hellman based puzzle scheme requires a modular exponentiation to obtain a correct solution. Note that a modular exponentiation can be divided into many modular multiplications [37] and a large number of modular additions. Our RSA Assumption-based scheme, on the other hand, only needs three modular multiplications and two additions to obtain a puzzle and its corresponding solution, which is more efficient.

- Our scheme is resistant to eavesdropping attacks. As we claimed earlier, no puzzle construction information is revealed during communication between the defending server and its clients. In addition, each connection request has a unique solution, so that eavesdropping is rarely advantageous to an attacker.

- By varying the values of public instances (such as M, e, N) and constant instances

(such as A, G_A, X_A), distinct web servers can obtain different puzzle schemes. This is more flexible for current network servers that possess various capabilities.

Through these remarks we conclude that our RSA Assumption-based puzzle scheme is qualified in theory.

5.5 Security Considerations

In this section, we will inspect the following possible attacks and show that our scheme holds strong security properties (under the RSA Assumption) against them.

(1) An adversary intends to predict the answer r_c from a sequence of valid answers that he obtained previously. Recall that

$$M = g^a \cdot r^e \pmod{N}$$

where M, N, e, g^a are the public parameters of a puzzle. The adversary then performs a brute-force search to find a valid answer r such that

$$r^e = \frac{M}{g^a} \pmod{N}.$$

He can also deduce the following equation, even without knowledge of d.

$$r = (\frac{M}{g^a})^d \pmod{N}$$

Similarly, he can also compute another answer

$$r' = (\frac{M}{g^{a'}})^d \pmod{N}.$$

Now, he can obtain a constant

$$\varphi = \frac{r}{r'} = (\frac{g^{a'}}{g^a})^d = (g^{a'-a})^d \pmod{N}.$$

Assume that

$$r'' = (\frac{M}{g^{a''}})^d \pmod{N}.$$

Then,

$$\begin{aligned} r'' \cdot \varphi &= (\frac{M}{g^{a''}})^d \cdot (g^{a'-a})^d \\ &= (\frac{M}{g^{a''+a-a'}})^d \pmod{N}. \end{aligned}$$

If the adversary can find that there is a relationship between the elements in A that satisfies $\hat{a} = a'' + a - a'$, he is able to pre-compute the answer

$$\widehat{r_c} = (\ \frac{M}{g^{\hat{a}}}\ )^d = (\frac{M}{g^{a''+a-a'}})^d = r'' \cdot \varphi \pmod{N}.$$

However, an important property of set A is that all the elements are derived from a superincreasing sequence. Hence, there is no such relationship.

(2) An adversary intends to reveal the secret constant y from the public parameters. However, he finds he has to compute the e-root of M (order N), which is equivalent to factorising pq.

(3) An adversary attempts to find out about set A from a sequence of public parameters $g^a, g^{a'} \ldots$ in the same period t. He knows that

$$g^a = g^{a_i} \cdot g^t \pmod{N}$$

$$g^{a'} = g^{a'_i} \cdot g^t \pmod{N}$$

then

$$\frac{g^a}{g^{a'}} = \frac{g^{a_i}}{g^{a'_i}} = g^{a_i - a'_i} \pmod{N}.$$

In fact, the adversary does not know whether the values of g^a and $g^{a'}$ come from the same period, because both t and g are secret parameters in our scheme. Even if we assume that t for g^a and $g^{a'}$ are uniform, the adversary cannot obtain any information about set A. Recalling the Discrete Logarithm Problem, we know that it is still hard to compute $(a_i - a'_i)$ even if g is known.

5.6 Summary

This chapter has depicted a novel client puzzle scheme embedded with an efficient trapdoor algorithm. This scheme has two distinct features: the computational overheads are low, and the difficulty level of puzzles is measurable. We also presented that this scheme is provably secure under the RSA Assumption.

Section 5.1 elaborated on a trapdoor algorithm, and proved that its security relies on the RSA Assumption. In Section 5.2, we implemented the Trapdoor-based Client Puzzle System (TCPS) by combining its definition (outlined in Chapter 4) with this trapdoor algorithm. In order to reduce the computational overheads of puzzle construction, we introduced a "Pre-construction" phase into our scheme. The major task

of "Pre-construction" is to pre-compute a sequence of constants that is required in puzzle construction and verification. We also demonstrated that puzzles produced by our proposed scheme satisfy all essential requirements and security conditions.

In this RSA Assumption-based TCPS, a defending server needs three modular multiplications and two additions to generate a puzzle along with a solution. A quick solution verification is achieved by only one value comparison. On the other hand, the complexity of puzzles can be controlled via an integer l, which is actually the length of solution search ranges. This means that the server can vary l to adjust the number of computations on the client's side, according to the strength of DoS attacks.

Section 5.3 provided a parameter table and a work prototype for this scheme. In Section 5.4, several possible attacks were considered. Under the RSA Assumption, our proposed scheme can defend against all these attacks.

In conclusion, this RSA Assumption-based TCPS does work effectively and efficiently.

Chapter 6

A Scheme Based On The DLP

This chapter will illustrate another trapdoor algorithm whose security is based on the hardness of Discrete Logarithm Problem (DLP). Similarly, we shall try to adapt this algorithm to the TCPS to fulfill puzzle construction. We shall maintain that this DLP-based scheme also meets the essential requirements and security conditions in the definition of the TCPS. In particular, compared with the RSA Assumption-based TCPS, this new scheme is expected to further diminish the cost of puzzle construction in two ways: one is in saving storage space, and the other one is in cutting down the number of modular multiplications.

6.1 Algorithm

The algorithm that we will introduce in this section is derived from Schnorr's Signature [12, 52], which was proposed in 1989 and could provide encryption and signature functions. Its security relies on the difficulty of computing discrete logarithms. First of all, we describe the algorithm as follows.

Let p be a large prime, and q be another prime which is the prime factor of $p-1$. The size of p and q should be selected with care to ensure that DLP in $\mathbb{Z}_p^*$ is hard. g is a generator of the group G (order q). Denote $x \in \mathbb{Z}_q^*$ as a positive and secret integer. Let l be the difficulty lever of puzzles.

Our scheme is based on the following equations, in which random $a,\ b \in \mathbb{Z}_q^*$.

$$h = g^x \pmod{p} \tag{6.1}$$

$$C = g^a \cdot h^b \pmod{p} \tag{6.2}$$

$$W = a + b \cdot x \pmod{q} \tag{6.3}$$

Examining the above equations, we find a transformation by assembling them together as shown below.

- Put Eq (6.1) into Eq (6.2).

$$\begin{aligned} C = g^a \cdot h^b &= g^a \cdot (g^x)^b \\ &= g^a \cdot g^{bx} \\ &= g^{a+bx} \pmod{p} \end{aligned} \tag{6.4}$$

- Put Eq (6.3) into Eq (6.4).

$$C = g^{a+bx} = g^W \pmod{p} \tag{6.5}$$

  Note that g, p are constant in Eq (6.5). Hence, if we can maintain the constancy of W, C should be a constant as well.

From equation (6.3), we can easily verify that a and b can be chosen arbitrarily, if the secret value x is known, to obtain W. Moreover, by a given random $a' \in \mathbb{Z}_q^*$, we can compute a unique $b' \in \mathbb{Z}_p^*$ by Eq (6.6) such that W is kept constant.

$$b' = (W - a') \cdot x^{-1} \pmod{q} \tag{6.6}$$

Our puzzle scheme operates so that using Eq (6.3) and (6.5), the puzzle creator can easily compute the initial values of W and C from a pair (a, b). Then, keeping them constant, the creator changes the value of a' for each new puzzle, and calculates a corresponding solution b' through Eq (6.6). He also generates a search range associated with b', where the range length equals l (recall that l is the difficulty level). The puzzle creator should broadcast p, h and Eq (6.2), but keep g, x, q, and W secret. In consequence, the only feasible method for a puzzle solver is to perform an exhaustive search in the candidate range to find a b' that makes Eq (6.2) hold.

Since he knows the trapdoor - Eq (6.6), the puzzle creator performs only one addition[1] and one modular multiplication to obtain a solution for each fresh a'. On the other hand, a solver must compute a sequence of modular exponentiations for testing each instance in the given range until he finds the correct answer. Meanwhile, the creator can adjust the size of the search range - l to control the computational costs consumed by the solvers.

[1] A subtraction operation can be viewed as an addition in a computer system.

Theorem 4 *If Eq (6.2) holds, then the solution b' computed by Eq (6.6) can be verified.*

Proof: Given the initial values of (a, b), we can obtain the following result.

$$C = g^a \cdot h^b = g^a \cdot g^{bx} = g^{a+bx} = g^W \pmod{p}$$

To create a new puzzle, we choose a' and compute the solution $b' = (W - a) \cdot x^{-1} \pmod{q}$. Now, we verify it in Eq (6.2).

$$\begin{aligned} C = g^W = g^{a'} \cdot g^{b'} &= g^{a'} \cdot h^{(W-a') \cdot x^{-1}} \pmod{p} \\ &= g^{a'} \cdot g^{(W-a') \cdot x^{-1} \cdot x} \pmod{p} \\ &= g^{a'} \cdot g^{W-a'} \pmod{p} \\ &= g^W \pmod{p} \end{aligned}$$

Theorem 5 *Our scheme is secure assuming the hardness of Discrete Logarithm Problem.*

Proof: To illustrate our proof, first let us recall the assumption made in the Discrete Logarithm Problem. Given two elements of a group, g^v and h, it is computationally infeasible to find a non-negative integer x such that

$$h = g^x \pmod{p}$$

holds, where the size of p is appropriately chosen (e.g. 1024 bits).

We assume there is an algorithm A that, given C, g, $h \in {Z_p}^*$, outputs a, b such that

$$C = g^a \cdot h^b \pmod{p}$$

holds with a non-negligible probability.

We will show an algorithm B that uses A to solve an instance of a Discrete Logarithm Problem which is assumed to be hard.

To be more precise, the task of algorithm B is to output

$$x = \log_g h \pmod{p}$$

given g, $h \in Z_p^*$ for a prime p. This simulation is as follows.

First, B provides g, h, p as the public parameters to A. Then, B performs the following:

- Select a random value $\theta \in Z_q^*$, where $q \mid p-1$.
- Compute: $C^* = g^\theta \bmod p$.

Finally, B provides C^* to A, and with a non-negligible probability, A outputs: (a, b) where:

$$C^* = g^a \cdot h^b \pmod{p}$$

holds. Now, the simulation is restarted. Again, first A is a provided with the public parameters g, h, p, and finally given C^*. A will produce another forgery $(\hat{a}, \hat{b})$ where

$$C^* = g^{\hat{a}} \cdot h^{\hat{b}} \pmod{p}$$

holds. If $\hat{a} = a$ and $\hat{b} = b$ hold, then the last simulation needs to be repeated until $\hat{a} \neq a$ and $\hat{b} \neq b$. We note that A will output $\hat{a} = a$ and $\hat{b} = b$ with probability $1/q^2$.

After obtaining two values (a, b) and $(\hat{a}, \hat{b})$, B gathers the following equations:

$$C^* = g^a \cdot h^b \pmod{p}$$

$$C^* = g^{\hat{a}} \cdot h^{\hat{b}} \pmod{p}$$

Hence, B derives:

$$g^a \cdot h^b = g^{\hat{a}} \cdot h^{\hat{b}} \pmod{p}$$

which implies:

$$a + bx = \hat{a} + \hat{b}x \pmod{q}$$

or

$$x \cdot (b - \hat{b}) = \hat{a} - a \pmod{q}$$

and

$$x = (\hat{a} - a)(b - \hat{b})^{-1} \pmod{q}$$

Note that x is the solution to the DLP problem. The probability of success of B is lower bounded by $1/q^2$, which is non-negligible. Hence, we have provided the proof.

6.2 A DLP-based TCPS

This section will present how a new TCPS works by using this DLP-based trapdoor algorithm. We still consider two participants in this scheme: a defending server S which can distribute puzzles for DoS defence; and a client C who is willing to pay

a countable computational cost to obtain service from S when it is suffering a DoS attack.

Similarly, we split the scheme into four phases, and use preprocessing to calculate and store a number of constants for puzzle construction and verification, which is expected to relieve the computational burden for S. Although there are some uniform parameters, such as T_d, t, which have been defined in Chapter 5, we still repeat them here to show this scheme in its entirety. An analysis of the efficiency and effectiveness of our scheme will be provided at the end of this section.

Now, we begin with the four phases.

1. **Pre-construction phase**

   (i) Determine a time interval T_d and the size of A denoted by n.

   Similar to the definition stated in Chapter 5, the relationship between T_d and n is a compromise. The server needs a minimum n and a maximum T_d to meet the following requirement: in each time period, n is by minimum greater than the maximum connection ability of the server. It means that no instance in set A can be chosen for puzzle construction more than once in same time period.

   (ii) Establish A.

   A is a random and non-repeated integer set.

$$A = \{a_i |\ a_i \in \mathbb{Z}_q^*, |\ 1 \leq i \leq n\} \cap \{\forall a_m, a_n \in A |\ a_m \neq a_n\}$$

   (iii) Compute G_A.

   G_A is computed by the server.

$$G_A = \{g^{a_i} \pmod p |\ 1 \leq i \leq n,\ a_i \in A\}$$

   (iv) Generate and encrypt t

   t is a time parameter which is in the form of

$$\{yymmddhh^1hh^2\} \cap \{hh^2 - hh^1 = T_d\}.$$

   We can adjust the form of t according to the designated T_d. For instance, if $T_d = 30$ minutes, the form of t can be described as

$$\{yymmddhhmm^1mm^2\} \cap \{mm^2 - mm^1 = 30\}.$$

   Then, the server computes $g^t \pmod p$ at the beginning of each new time period.

(v) Compute initial C, W, w_t and $x^{-1} \pmod q$.

Given a random pair $(a, b) \subset Z_q^*$, the server obtains

$$W = a + b \cdot x \pmod q$$

and

$$C = g^W \pmod p.$$

Compute

$$w_t = W - t \pmod q.$$

Using the extended Euclidean algorithm, the server easily computes $x^{-1} \pmod q$.

In the pre-construction phase, the server obtains constants $\{W, C, x^{-1} A, G_A\}$ and two periodic constants $\{g^t, w_t\}$.

2. **Construction phase**

When receiving a request from a client C, the server S generates a puzzle $\mathcal{P} = (g^{a'}, [\alpha, \beta])$ along with its correct solution b_s. $\mathcal{P}$ is essential for a client solving a puzzle. b_s is stored on the server for verification of the answer returned from the client. This stage is described as follows.

(i) Compute $g^{a'}$

$$g^{a'} = g^t \cdot g^{a_i} \pmod p \tag{6.7}$$

where $g^{a_i} \in G_A$, t is the current time period and g^t is a periodic constant calculated at the beginning of t. The server picks up random $g^{a_i} \in G_A$ to generate a fresh puzzle, then marks this g^{a_i} to be unavailable until a new period comes. This means that when each period starts, all the elements in G_A are available. An element is marked unavailable throughout the same period, once it is chosen by the server to create a puzzle.

(ii) Generate b_s

$$\begin{aligned} b_s &= (W - a') \cdot x^{-1} && \pmod q \\ &= [W - (t + a_i)] \cdot x^{-1} && \pmod q \\ &= (w_t - a_i) \cdot x^{-1} && \pmod q \end{aligned} \tag{6.8}$$

Note that the values of i in Eq (6.7) and Eq (6.8) are uniform.

(iii) Obtain a search range $[\alpha, \beta] \subset \mathbb{Z}_q^*$

$$\alpha = b_s - c \pmod{q}$$

$$\beta = \alpha + l \pmod{q}$$

for random $c \in [0, l)$, where l is the current difficulty level of the puzzles.

3. **Puzzle-solving phase**

Unlike the other three, this phase is performed on the client's side. We assume that a client $\mathcal{C}$ has installed a specific piece of software which is distributed by the server. It solves puzzles received from the server by using a fixed equation

$$C = g^a \cdot h^b \pmod{p},$$

a triple of constants (C, h, p), and an interface for accepting puzzle parameters from the server.

When a client receives a puzzle $\mathcal{P}$, he employs an exhaustive search (brute-force) to find the answer b_c which satisfies the equation. Due to the length of the search range $l = \beta - \alpha$, a client $\mathcal{C}$ needs to perform on average $l/2$ modular exponentiations to find the answer b_c.

4. **Verification phase**

Upon receiving the answer b_c from a client, the server compares it with the stored solution b_s that has been calculated in the construction phase. If they are equal, **Verify**$(\cdot)$ outputs 1, which means that authentication of the client is verified, and the server proceeds with the rest of the request. Otherwise, **Verify**$(\cdot)$ outputs 0, and the server drops the request.

$$\textbf{Verify}(b_s, b_c) = \begin{cases} 1 & \text{if } b_s = b_c \\ 0 & \text{otherwise} \end{cases}$$

6.3 Parameter Table and Scheme Prototype

No.	Notation	Phase	Property	Public/Private	Description
1	l	1	(*)Const.	Private	difficulty level of puzzles
2	p	1	Const.	public	large prime
3	q	1	Const.	Private	$q \vert p-1$
4	x	1	Const.	Private	random $x \in \mathbb{Z}_q^*$
5	g	1	Const.	Private	g is a generator in Z_q^*
6	h	1	Const.	Public	$h = g^x \pmod p$
7	a	1	Random	-	initial value
8	b	1	Random	-	initial value
9	C	1	Const.	Public	$C = g^a \cdot h^b \bmod p$
10	W	1	Const.	Private	$W = a + bx \pmod q$
11	T_d	1	Const.	Private	time interval
12	t	1	(*)Const.	Private	$hh^2 - hh^1 = T_d$
13	g^t	1	(*)Const.	Private	-
14	w_t	1	(*)Const.	Private	$w_t = W - t \pmod q$
15	x^{-1}	1	Const.	Private	-
16	A	1	Const.	Private	$a_i \in I$
17	G_A	1	Const.	Private	$g^{a_i} \bmod N$
18	$g^{a'}$	2	Var.	Private	$g^{a'} = g^t \cdot g^{a_i} \pmod p$
19	b_s	2	Var.	Private	Server's solution
20	r	2	Random	Private	-
21	α	2	Var.	Public	$\alpha = r_s - r$
22	β	2	Var.	Public	$\beta = \alpha + l$
23	b_c	3	Var.	Public	Client's answer

Table 6.1: Parameter Table of the DLP-based TCPS

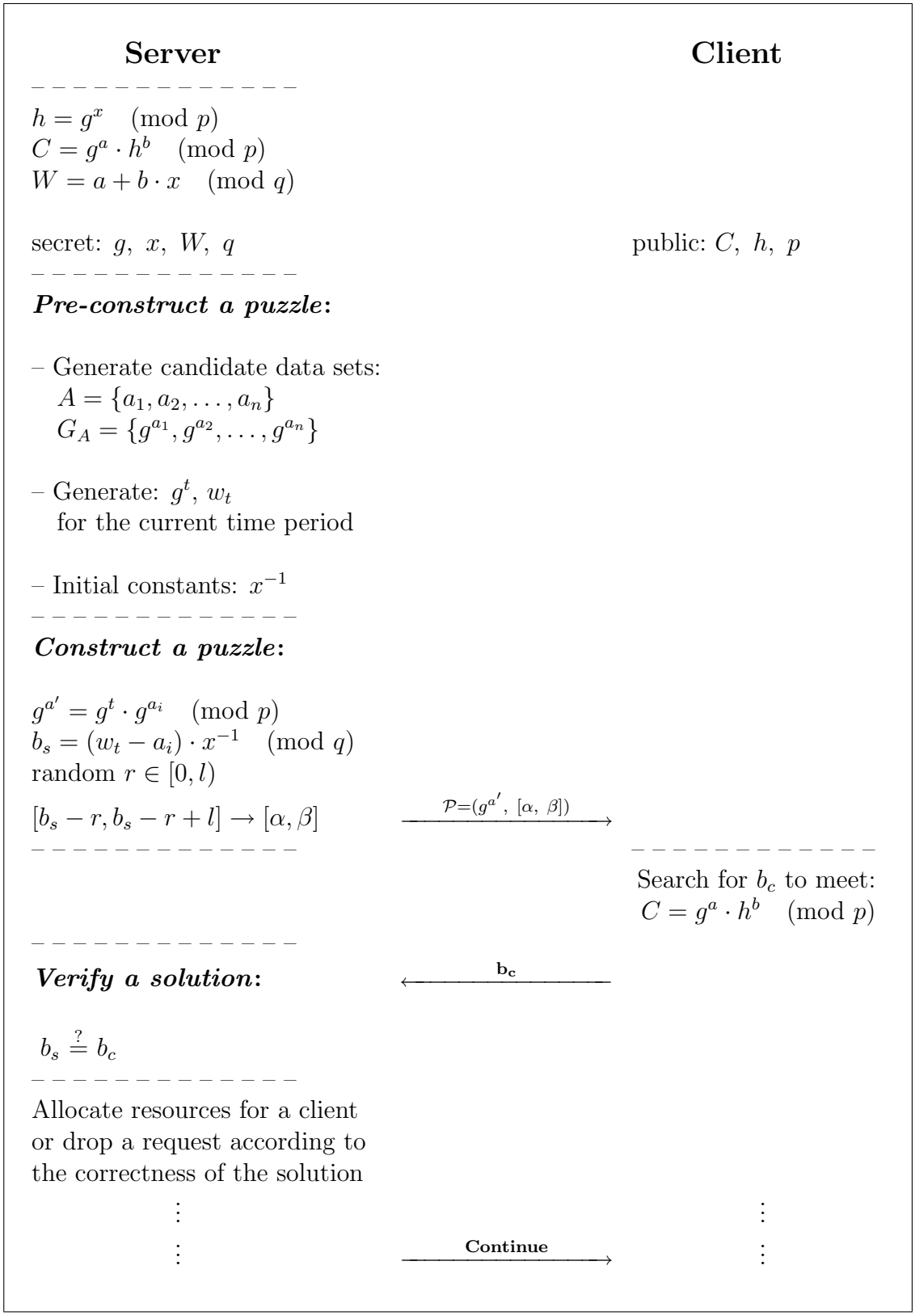

Figure 6.1: A DLP-based Puzzle Scheme

6.4 Remarks

Theorem 6 *The DLP-based puzzle scheme will produce a Trapdoor-based Client Puzzle System.*

Proof. We need to show that this DLP-based puzzle scheme meets all the security requirements as outlined in the definition of the TCPS.

- *Correctness.* It is easy to verify that answers computed correctly in the puzzle-solving phase can always pass the verification.

  In the proposed scheme, we provide a search range $[\alpha, \beta]$ which ensures the existence of the correct solution b_s. Because

  $$[\alpha, \beta] \leftarrow [b_s - r, b_s - r + l] \quad for\ r \in [0, l)$$

  where both r, l are positive integers, this means

  $$b_s - r < b_s < b_s - r + l$$

  Hence, a client can eventually find a b_c $(= b_s)$ which makes Eq (5.1) valid, as long as he employs an exhaustive search.

- *Semantic Security.* Due to the hardness of the Discrete Logarithm Problem, a more efficient way to obtain the correct answer than a brute-force search cannot be found. We will prove this claim in the Security Considerations section below.

- *Collision-resistance.* According to the property presented in Eq (6.2), for each $a' \in \mathbb{Z}_p^*$, there exists a unique value $b' \in \mathbb{Z}_q^*$ which can satisfy the equation $C = g^{a'} \cdot h^{b'} \bmod p$.

- *Information Theoretic Secrecy.* In the pre-construction process, the defending server generates a random and non-repeatable data set A ($A = \{a_i | 1 \leq i \leq n,\ a_i \in Z_q^*\}$). According to set A, the server obtains a multiplier factor set G_A ($G_A = \{g^{a_i} \pmod p \mid 1 \leq i \leq n,\ a_i \in Z_q^*\}$) in which the values of g^{a_i} are consequently irregular.

  When the defending server constructs a puzzle, it performs the following computation:

  $$g^{a'} = g^t \cdot g^{a_i}$$

where g^{a_i} is randomly selected from G_A. We notice that the probability of a client successfully predicting a puzzle equals $1/qn$. When q and n are big enough, pre-computing a puzzle is computationally infeasible in the time period t.

This scheme also meets the essential requirements that we advocated in Chapter 4 for promising client puzzles.

Remark 3 *The computational resources consumed by a client to solve a puzzle are greater than those used by the server to generate a puzzle and verify its solution.*

A defending server performs two modular multiplications and three additions to create a puzzle and obtain the solution.

To produce $g^{a'}$, the server performs one modular multiplication:

$$g^{a'} = g^t \cdot g^{a_i} \pmod{p}$$

To obtain the solution, the server performs one modular multiplication and one addition:

$$b_s = (w_t - a_i) \cdot x^{-1} \pmod{q}$$

where w_t is computed at the beginning of the current time period.

To generate the search range $[\alpha, \beta]$, the server performs two additions:

$$\alpha = b_s - r$$

$$\beta = \alpha + l < N$$

To verify a solution, the server only needs to make a comparison.

$$b_s \stackrel{?}{=} b_c$$

Since we ensure that the size of p and q are large enough to resist any attacks on the DLP, the answer can only be obtained by exhaustively searching the seed range and performing modular exponentiations until the correct instance that satisfies the public equation is found. In consequence, to solve a puzzle, a client has to conduct on average $l/2$ modular exponentiations and comparisons, which is much more than the defending server does.

Remark 4 *This DLP-based puzzle scheme is better than the Hash function [5] and the Diffie-Hellman [6] based schemes, and also even better than the RSA Assumption-based puzzle scheme.*

Similarly, the DLP-based TCPS is more efficient and effective than most proposed puzzle schemes.

- A quick verification is achieved by using only one comparison, which relies on the uniqueness of puzzle solutions.
- The defending server can adjust the complexity of client puzzles by changing the value of l. Moreover, the defending server is able to predict the average computational cost on the client's side.
- Most constant instances computed in the pre-construction phase can be reused in each time period, which reduce the workload of the defending server.
- Like the RSA Assumption-based puzzle scheme, this new method is also resistant to eavesdropping attacks.
- Compared to the RSA Assumption-based puzzle scheme, this DLP-based scheme has two advantages:
  - Reducing computational costs. The DLP-based scheme only needs two modular multiplications and three modular additions, whilst the RSA Assumption-based one requires three modular multiplications and two modular additions.
  - Saving memory space. In the pre-construction phase, the DLP-based scheme only needs two constant sets A and G_A. The RSA Assumption-based one, on the other hand, requires three constant sets A, G_A and X_A.

Following the above remarks, we conclude that the DLP-based puzzle scheme is qualified in theory.

6.5 Security Considerations

In this section, the proposed digital signatures will be analysed by inspecting the following possible attacks. We show that if an adversary intends to break our puzzle scheme, he has to solve the the discrete logarithm problem in $\mathbb{Z}_p^*$, which is expected to be hard.

- ***Scenario 1***. An adversary attempts to find the secret integer x from the public parameters h and p, since $h = g^x \pmod p$. The adversary finds that even though he knows g[2], he still has to compute a discrete logarithm in $\mathbb{Z}_p^*$.

- ***Scenario 2***. An adversary intends to find the secret constant W from the public parameter C, p, since $C = g^W \pmod p$. He finds a similar situation with Scenario 1. He has to compute a discrete logarithm in $\mathbb{Z}_p^*$, even though he knows g.

- ***Scenario 3***. An adversary intends to find secret information about the puzzle construction a' from the public parameter $g^{a'}$. This is still hard because it is also a typical discrete logarithm.

6.6 Summary

This chapter has described another trapdoor algorithm, which is also suitable for the TCPS. This new client puzzle system is secure assuming the hardness of the DLP.

Section 6.1 elaborated on this trapdoor algorithm, and proved that its security relies on the Discrete Logarithm Problem. In Section 6.2, we implemented the Trapdoor-based Client Puzzle System (TCPS) by combining its definition (outlined in Chapter 4) with this DLP-based trapdoor algorithm. Similar to the RSA Assumption-based puzzle scheme, there are four phases in this new scheme, which are "Pre-construction" phase, "Construction" phase, "Puzzle-solving" phase and "Verification" phase respectively.

Compared with the RSA Assumption-based scheme, this DLP-based TCPS has two more major advantages: reducing computational costs by using two modular multiplications and three additions to construct a puzzle (the RSA Assumption-based one requires three modular multiplications and two additions); saving memory space in the pre-construction phase by calculating and storing only two constant sets (the RSA Assumption-based one requires three constant sets).

We also demonstrated that puzzles produced by this DLP-based scheme satisfy all essential requirements and security conditions.

In conclusion, this DLP-based TCPS is qualified in DoS defences.

[2] g is also kept secret in this scheme.

Chapter 7

System Description and Discussion

In this chapter, we shall describe the working environment for our Trapdoor-based Client Puzzle System (TCPS), and how a defending server will work together with the TCPS. We shall provide a coarse-gained configuration for system parameters, which will enable the server to detect DoS attacks and start up the defence system in time. Moreover, we will give an example of how to determine the difficulty level of puzzles according to the load on the defending server. After that, we shall discuss several possible attacks against the TCPS, and further solutions will be analysed.

7.1 Working Environment

To imitate the external environment surrounding the TCPS, we create an attack model and assume that there are four basic components in a network: a defending server deploying a client puzzle protection, a number of normal clients who are willing and able to pay some computational costs for quality service when the server is suffering from a DoS attack, a malicious client (or an adversary) with limited computational resources who is attempting to launch a Distributed Denial-of-Service attack, and some unwitting accomplices whose machines are called "Zombies" and serve as a platform for a DoS attack. To provide service, the defending server and a client communicate with each other under a certain network protocol, in which our client puzzle mechanism is embedded.

We make the following assumptions about the capabilities of the adversary and the "Zombies". The first three have been explained by Juels and Brainard [33], and are included in the following list:

1. The adversary cannot modify packets sent from any client to the defending server.

2. The adversary cannot significantly delay the packets sent from any client to the defending server.

3. The adversary and the "Zombies" can exploit spoofed IP addresses to launch a DoS attack.

4. "Zombies" can comply with the adversary's command to launch cooperated attacks at a designated time, but they cannot help the adversary to compute puzzles.

5. An adversary can deploy network worms to recruit a cohort of "Zombie" machines in the widespread network. However, he/she cannot generally compromise routers in the middle of the network. Based on this assumption, we consider that an adversary can only perform limited eavesdropping on the network.

4. All client machines have the same processing power to devote to puzzle solving, and we view an adversary as a compromised client machine. Since our client puzzle system is deployed in conjunction with conventional time-outs, only the solution returned before its expiration can be accepted by the defending server. Consequently, a legitimate client can easily compute a single puzzle and send it back in time, whereas an adversary must have access to large amounts of computational resources to achieve his goal. Due to the fourth assumption above, an adversary cannot gather such large amounts of resources on his own.

7.2 System Description

We shall now look at how a system using our trapdoor-based client puzzle works for a defending server and legitimate clients. Our scheme begins with a regular examination of a defending server to determine whether it is currently under a DoS attack. A standard parameter for an attack can be the status of buffer space consumed or the number of TCP connections. Here, we prefer the status of buffer space consumed as a standard; so let B_f denote the whole buffer space of a defending server and C_{sm} be the number of buffers consumed. We assume there are five values of C_{sm} which reflect the strength of an attack (this can be adjusted flexibly according to different web servers), and we call them 1 to 5. When less than half of B_f is consumed, the value of C_{sm} is zero, which means there is no attack alarm for a defending server. When the number of consumed buffers comes to half of B_f, the value of C_{sm} is increased to 1, indicating

that the server is suspected of being under an attack. From now on, the defending server needs to construct and send out a puzzle for every client who is applying for a connection until the alarm is removed again. When the number of consumed buffers increases to 3/5 B_f, 7/10 B_f, 4/5 B_f, and 9/10 B_f respectively, the values of C_{sm} are 2, 3, 4, and 5 correspondingly.

We have another parameter l – the difficulty level of a puzzle, which decides the length of the search range related to C_{sm}. When C_{sm} is zero, the value of l is also zero. When C_{sm} is increased to 1, l becomes 10 (the value of l can also be adjusted according to different web servers). As C_{sm} increases, the value of l appears to grow exponentially. We describe some variations below:

$$
\begin{aligned}
&C_{sm} = 0, \qquad l = 0;\\
&C_{sm} = 1, \qquad l = 10;\\
&C_{sm} = 2, \qquad l = 100;\\
&C_{sm} = 3, \qquad l = 1,000;\\
&C_{sm} = 4, \qquad l = 10,000;\\
&C_{sm} = 5, \qquad l = 100,000;
\end{aligned}
$$

We can see that the difficulty of a puzzle can be increased from 0 to a figure that is computationally infeasible as an attack becomes more severe.

We describe the details of our defending system as follows.

1. $C_{sm} = 0$**, which means there is no attack alarm for our defending server.**

   When a client sends an initial connection request and an enquiry as to whether the server is under a DoS attack, the server answers "No". Then the client and the server should continue and finish their connection according to a standard connection establishment protocol such as TCP. The period of validation for this connection is within a certain time T_v.

2. $C_{sm} \geq 1$**, which means our defending server is suspected of being under a DoS attack**

   When a client sends an initial connection request and an enquiry as to whether the server is under a DoS attack, the defending server constructs a puzzle according to the current value of l, and sends back "Yes", together with a set of puzzle parameters, to the client. The server keeps a correct solution b'_{si} for every connection request i for a specified period T_w. A client uses the puzzle's parameters

to solve the problem and sends the solution back. The defending server compares the two solutions, and then decides whether to proceed or drop this request. A connection request should be dropped immediately if a solution received from a client is incorrect, or the time for sending back a solution is beyond T_w.

7.3 Discussion

In this section, we will discuss some possible attack scenarios and problems that occur when a defending server implements the TCPS. Solutions will also be discussed.

- **Discussion 1**

  If a malicious client floods a defending server with initial connection requests by using spoofed IP addresses or his "Zombies" (see Figure 6), how can a server protect itself? Wasting system resources to construct and send out a puzzle for every connection request is not an efficient approach for protecting a web server.

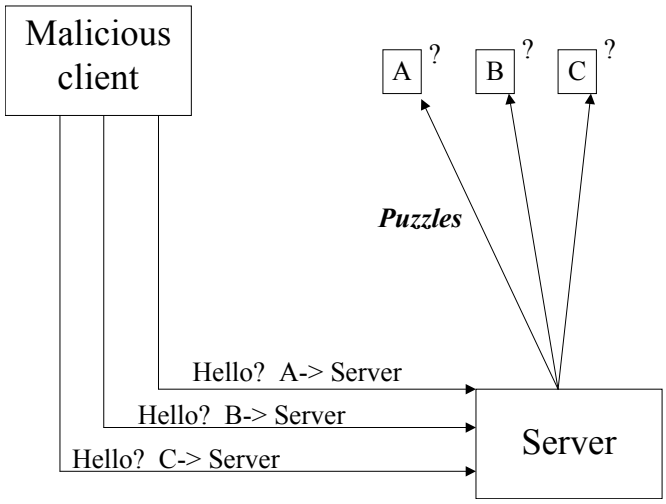

Figure 7.1: A Threat To Our TCPS

The aim of such an attack is to make a defending server perform a large number of meaningless computations for the construction of puzzles, which may lead to the possibility of exhausting the server's resources. Recall that a basic principle of client puzzles in defending against DoS attacks is to force a client to consume a large number of resources before obtaining resources from the defending server. If we do not comply with this rule, the threat of DoS attacks still exists.

We propose two improved solutions to defeat this threat.

Solution 1

We propose to calculate and obtain the puzzle solution pairs $(g^{a'}, b'_s)$ in advance, and store them in a secret table on a defending server, before every new time period t starts. When there is a connection request, the server does not need to construct a puzzle by calculating the modular multiplication for $g^{a'}$ and b'_s. The server just fetches a pair of puzzle solutions $(g^{a'}, b'_s)$ from the table, keeps b'_s, and sends the value of $g^{a'}$ to the client. Hence, the only thing that the server needs to do, when there is a connection request, is to select a random integer r to generate the seed range $[b'_s - r, b'_s - r + l]$ for b'_c. The size of the table can be determined by the capability of the connectivity of the defending server.

Solution 2

Before describing our second scheme, we make the following assumption for this solution. A malicious client who deploys IP spoofing cannot intercept all the packets sent from a defending server to the spoofed addresses. When a defending server receives a connection request, it sends back a sequence number to determine whether the address is bogus. The aim of this option is to alleviate an attack via the sequence number in this first step, and then ban it in the later puzzle verification.

When a client sends an initial connection request and an enquiry as to whether the server is under a DoS attack, the server returns "Yes" and a sequence number to the received IP address. If the client is willing to accept the puzzle, he should return a value that is equal to the sequence number plus one in a strict time period T_s. After receiving the correct sequence number in a valid time period, the server generates and sends a cryptographic puzzle to the client. Otherwise, the server will drop the service request immediately. The sequence number is a filter for spoofed IP addresses which works under the above assumption. A legitimate client has to simply solve a puzzle and send the solution in a given time T_w to obtain the final service. If the solution is correct, the server will proceed with the rest of the connection request and distribute the system's resources to the client. On the other hand, if the answer is wrong or the time for computation is beyond the given time T_w, the server will drop this request. If a connection is established, the period of validation for this connection is within a specified time T_v.

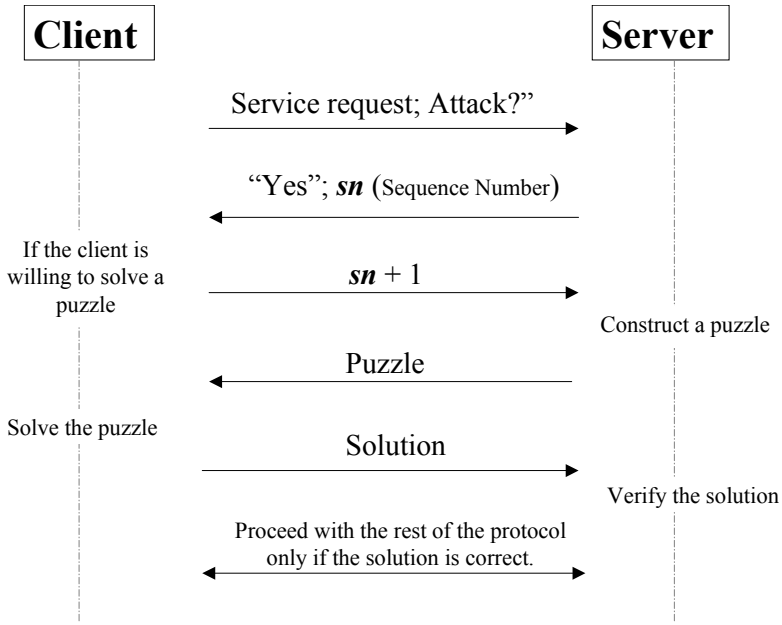

Figure 7.2: Using A Sequence Number Against IP Spoofing

- **Discussion 2**

  Compared with a hash-based puzzle, in which a defending server does not need to store any client information while a client is solving the puzzle, our scheme requires a defending server to store a solution b'_s. These are two totally different purposes. The hash approach sacrifices CPU time in the verification process to avoid any memory becoming exhausted. Our trapdoor-based puzzle conducts a reverse activity. We note that the final decision should be made by the server administrators, who are aware of which resources are more valuable for their own systems.

- **Discussion 3**

  The main goals of our client puzzle schemes are to defend network servers against DoS attacks, and ensure that legitimate clients gain qualified service as long as they can solve the puzzles. However, some DoS attacks may aim to take down clients' machines, where an attacker may broadcast false puzzles to consume a client's resources.

  In this case, the current solution (for example, the puzzle scheme proposed by Aura [5]) is to make puzzles signed by the defending server. When receiving a

puzzle, a client first verifies the signature by using the public key of the defending server, and then computes it. Since the number of puzzles for a legitimate client is very small, the cost of public-key verification on the client's side is usually acceptable.

- **Discussion 4**

  Our TCPS and two proposed puzzle schemes can be applied to all network servers which are based on TCP connection, such as telnet, Web or email, or SSL/TLS [21]. The software for puzzle construction and computation can be homogenous and centrally distributed. Different servers can obtain diverse puzzle systems by changing a series of parameters in the schemes, for example, the large primes (e.g. $p,\ q$), the constant sets (e.g. $A,\ G_A$), and so on. Clients need to download the software only once, and are then able to solve all types of puzzles, according to parameters and equations published by servers.

Chapter 8

Conclusion and Future Work

This thesis began with a brief introduction about the achievements of today's Internet, along with an overview of the major security threats to its development. Our work in this thesis, however, has only focused on studying and preventing one of these threats: Denial-of-Service (DoS), which is believed to create massive destruction in open communication networks.

By reviewing the history of DoS, we have acknowledged that Denial-of-Service is a macro definition for the result of all intentional/accidental actions that can cause service to be degraded or interrupted. A DoS attack is specifically referred to in our work as a malicious action which attempts to render an Internet service provider incapable of supplying normal service to legitimate clients.

To illuminate the attack theory, we have demonstrated a number of representative examples of DoS attacks, in which Distributed Denial-of-Service (DDoS) attacks are relatively new and possess more significant power by controlling "Zombie" armies and mounting cooperated attacks on victim hosts or networks. We have classified these attacks in two distinct ways according to their characteristics.

Several proposed countermeasures have been analysed, such as Ingress/Egress Filtering and Packets Marking. An analysis of their advantages and limitations has been presented in Chapter 2. Compared with these methods, client puzzles appear to be more reasonable and more practical to protect web servers against resource depletion attacks. We have described the robustness of client puzzles, which force suspected attackers (including all legitimate clients and real attackers) to invest their computational resources in an authentication process before a defending server allocates its memory and processing time.

However, we have also found that most proposed puzzle constructions are incapable of meeting the demands of tackling large amounts of high-speed incoming requests. In

the most severe cases, the client puzzle mechanism itself may become a target of DoS attacks. By examining two proposed puzzle schemes, we have summarised current problems and provided our recommendations.

Since puzzle distribution and verification can be subject to DoS attacks, we have proposed a novel model for puzzle construction, called the Trapdoor-based Client Puzzle System (TCPS). As long as it satisfies the security conditions we have defined, any cryptographic algorithm can be adapted to this model to construct puzzles. Our TCPS can overcome the problems encountered by previous puzzle schemes, such as the complexity of construction and verification, solution collision, and so on.

Moreover, we have developed two puzzle schemes in this thesis, and demonstrated that both of them are efficient with low computational costs, and can be deployed in connection with the Client Puzzle Protocol [33] or other existing network protocols (e.g. TCP/IP, SSL). We have also provided complete security proof for each respective scheme.

The common attributes of our new puzzle schemes are outlined as follows:

- The defending server sends the client a puzzle whose solution requires a brute-force search and a number of modular exponentiations to satisfy a given equation.
- The difficulty level of puzzles is parameterised according to the server load. In addition, the average computational costs on the client's side can be predicted by the server.
- The security of both puzzle schemes is guaranteed by some hard problems in mathematics, such as the Factorisation and the Discrete Logarithm.
- Most of the computations can be accomplished in the pre-construction phase. Due to the special cryptographic properties of puzzles, pre-computed items can be reused in connection with time parameters, and newly generated puzzles still appear random.
- Most significantly, a quick verification for puzzle solutions can be achieved, which relies on the uniqueness of solutions. The server is able to obtain a unique solution when a puzzle is created, then compare it with the answer received from the client. Hence, there is no computation in the verification phase.

After describing two puzzle schemes, we have proposed a configuration of system parameters to help the defending server detect DoS attacks and start up the client

puzzle protection. A discussion on further attacks was presented, and we noted that while client puzzles are promising in terms of being able to mitigate the damage of DoS attacks, a large number of research issues are still waiting to be resolved in the future.

Future Work

The TCPS introduced in this thesis is a first step towards setting up a framework to solve the problems that exist in current puzzle constructions. More cryptographic algorithms are expected to be found and adapted to this model. The results of our two specific puzzle schemes have been theoretically proved in this thesis. It would be exciting to come up with a real system in which we can determine their effectiveness through experiments.

In addition, as analysed in Chapter 2, many reasons contribute to the prevalence of DoS/DDoS attacks today, which include both network problems and individual problems. Client puzzles are designed to improve the Internet's protocol, which aims at remedying network defects. Other solutions should arise from effective Intrusion Detection Systems (IDS) or antivirus systems to tackle individual problems. Currently, client puzzles can only mitigate DoS attacks, but cannot stop them. We suggest combining puzzle schemes with other countermeasures to achieve this goal in the future.

Another interesting issue is that our work in this thesis focuses only on how to defend network servers against resource depletion attacks by using client puzzles. In fact, this mechanism can also be used to defeat network congestion or bandwidth depletion attacks by asking a sender to solve puzzles if he/she intends to send out a large traffic load. However, few people have noticed this topic to date, and it still is an open research question.

In conclusion, we summarise possible future work as follows:

- Implement the two puzzle schemes proposed in this thesis and perform experiments on their effectiveness.
- Study how to control the complexity of client puzzles to maximise a defending server's unitisation in practice.
- Cooperate with other countermeasures, such as Packet Marking or Traceback IP, to trace suspected packets to their source IP addresses. Finally, identify and remove "Zombies", "Masters", and even true attackers.

- Extend the applications of client puzzles to relevant security fields.

The concept of client puzzles is a relatively new topic in network security, and therefore few implementations have so far been proposed. We hope that our research has provided some valuable recommendations, which will be beneficial to other researchers in this field.

Bibliography

[1] Digital signature standard (DSS). In *Federal Information Processing Standards Publication 186*. National Institute of Standards and Technology (NIST), 1994.

[2] The New York Times, 12 September, 1996.

[3] R. Aguilar and J. Kornblum. New York Times site hacked. CNET NEWS.COM, 8 November 1996.

[4] T. Aura and P. Nikander. Stateless connections. In *Proceedings of International Conference on Information and Communications Security ICICS'97*, volume 1334 of LNCS, pages 87–97. Springer Verlag, November 1997.

[5] T. Aura, P. Nikander, and J. Leiwo. Dos-resistant authentication with client puzzles. *Security Protocols, 8th International Workshop, Cambridge, UK, April 3-5, 2000; revised papers*, volume 2133 of Lecture Notes in Computer Science:170–177, Springer 2001.

[6] J. A. Halderman B. Waters, A. Juels and E. W. Felten. New client puzzle outsourcing techniques for dos resistance. In *ACM Conference on Computer and Communications Security*, pages 246–256, 2004.

[7] B. Bencsth, I. Vajda, and L. Buttyn. A game based analysis of the client puzzle approach to defend against dos attacks. In *IEEE Conference on Software, Telecommunications and Computer Networks (SoftCom 2003)*, Venice, 7-10 October, 2003.

[8] D. J. Bernstein. Syn floods - a solution.
Available at http: // www. op. net/ jaw/ syn-fix.html, 1996.

[9] Matt Bishop. *Computer security : art and science*. Boston, MA : Addison-Wesley, 203.

[10] D. Boneh. Twenty years of attacks on the rsa cryptosystem. *In Notices of the American Mathematical Society (AMS)*, 46:203–213, 1999.

[11] R. Braden. T/tcp: Tcp extensions for transactions functional specification. *RFC 1644*, July 1994.

[12] E. F. Brickell and K. S. McCurley. An interactive identification scheme based on discrete logarithms and factoring. In *Advances in Cryptology, Proc. EUROCRYPT 90,* LNCS 473, volume 5, pages 23–29. Springer-Verlag, 1991.

[13] CIAC Information Bulletin. H-02: SUN's tcp syn flooding solutions, October, 1996.

[14] CERT Coordination Center. Trends in denial of service attack technology, October, 2001.

[15] Wu chang Feng. The Case for TCP/IP Puzzles. In *Proceedings of ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA-03)*, Germany, August 2003.

[16] CNN. Cyber-attacks batter Web heavyweights.
available at http://www.cnn.com/2000/tech/computing/02/09/cyber.attacks.01/index.html, February 2002.

[17] R. Cramer and I. Damgard. New generation of secure and practical rsa-based signatures. In *Advances in Cryptology, Proc. CRYPTO 96,* LNCS 1109, pages 173–185. Springer-Verlag, 1996.

[18] R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. *ACM Transactions on Information and System Security*, 3(3):161-185, August 2000.

[19] daN. Re: client puzzle protocol neohapsis archives.
Available at http://archives.neohapsis.com/archives/nfr-wizards/2000-q1/0645.html, 2000.

[20] C. Davidson. The "SYN flood" gates open for WebCom. iWorld Weekly, 16 December 1996.

[21] D. Dean and A. Stubblefield. Using client puzzle to protect TLS. In *Proceedings of the 10th USENIX Security Symposium*, USA, August, 2001. USENIX Association.

[22] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology, Proc. CRYPTO 92,* LNCS 740, pages 139–147, Santa Barbara, CA USA, August 1992. Springer Verlag.

[23] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985.

[24] J. Elliot. Distributed denial of service attacks and the zombie ant effect. *IT Professional*, pages 55–57, March, 2000.

[25] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing. *IETF, RFC 2267*, January 1998.

[26] M. K. Franklin and D. Malkhi. Auditable metering with lightweight security. *Journal of. Computer Security*, 7(4):237–255, 1998.

[27] A. Freier, P. Karlton, and P. Kocher. The SSL protocol -version 3.0, March 1996.

[28] X. Geng, Y. H., and A. B. Whinston. Defending wireless infrastructure against the challenge of ddos attacks. *Mobile Networks and Applications (MONET)*, 7(3):213–223, 2002.

[29] R. Gennaro. Multitrapdoor commitments and their applications to proofs of knowledge secure under concurrent man-in-the-middle attacks. In *Advances in Cryptology, Proc. CRYPTO 2004,* LNCS 3152. Springer-Verlag, 2004.

[30] Thomer M. Gil. MULTOPS: a data-structure for bandwidth attack detection, 2001.

[31] Shon Harris. Dos defense. *Information Security magazine*, September, 2001.

[32] John D. Howard. An analysis of security on the internet 1989 - 1995, PhD thesis, Carnegie Mellon University, April 1997.

[33] A. Juels and J. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In S. Kent, editor, *Distributed Systems Security (SNDSS)*, pages 151–165, 1999.

[34] F. Kargl, J. Maier, and M. Weber. Protecting web servers from distributed denial of service attacks. In *Proceedings of the 10th International WWW Conference*, Hong Kong, May 1-5, 2001.

[35] Charlie Kaufman, Radia Perlman, and Mike Speciner. *Network Security: Private Communication in a Public World (2nd Edition)*. Prentice Hall PTR, 2002.

[36] A. K. Lenstra and H. W. Lenstra Jr. Algorithms in number theory. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 673–715. MIT Press/Elsevier, 1990.

[37] C. Mclvor, M. Mcloone, and J. V. Mccanny. Modified montgomery modular multiplication and rsa exponentiation techniques. In *IEE Proceedings Computers & Digital Techniques*, volume 151, pages 402–408, Nov 2004.

[38] R. C. Merkle. Secure communications over insecure channels. *Communications of ACM*, 21(4):294–299, April 1978.

[39] P. R. J. Mirkovic and J. Martin. A taxonomy of ddos attacks and ddos defense mechanisms. *Technical Report 18, University of California, Los Angeles - Computer Science Department*, 2002.

[40] A. M. Oldyzko. Discrete logarithms in finite fields and their cryptographic significance. In *Advances in Cryptology, Proc. EUROCRYPT 84,* LNCS 209, pages 224–314. Springer Verlag, 1984.

[41] T. J. Ott, T. V. Lakshman, and L. Wong. Sred: Stabilized red. In *Proceedings of IEEE INFOCOM, March 1999*, pages 1346–1355, 1999.

[42] K. Park and H. Lee. On the effectiveness of probabilistic packet marking for ip traceback under denial of service attack. *IEEE INFOCOM 2001*, pages 338–347, 2001.

[43] K. Park and H. Lee. On the effectiveness of route-based packet filtering for distributed dos attack prevention in power-law internets. In *Proceedings of ACM SIGCOMM'2001*, August 2001.

[44] K. Park and H. Lee. Advanced packet marking mechanism with pushback for ip traceback. In *ACNS04 PROGRAM Academic Track*, June 8-11, 2004.

[45] R. C. Peralta. A simple and fast probabilistic algorithm for computing square roots modulo a prime number. *IEEE Transactions on Information Theory*, IT-32:846–847, 1986.

[46] DARPA INTERNET PROGRAM. RFC 791 - internet protocol, September, 1981.

[47] DARPA INTERNET PROGRAM. RFC 793 - transmission control protocol, September, 1981.

[48] Michael B. Rash. client puzzle protocol.
Available at http://honor.trusecure.com/pipermail/firewall-wizards/2000-february/007944.html, 2000.

[49] L. Ricciulli, P. Lincoln, and P. Kakkar. TCP SYN flooding defense. In *In Comm. Net. and Dist. Systems Modeling and Simulation Conf. (CNDS' 99)*, 1999.

[50] R. Rivest, A. Shamir, and L. Adelman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, February 1978.

[51] Bruce Schneier. *Applied cryptography : protocols, algorithms, and source code in C.* John Wiley & Sons, Inc., 1996.

[52] C. P. Schnorr. Efficient signature generation for smart cards. In *Advances in Cryptology, Proc. CRYPTO 89,* LNCS 435, pages 239–252. Springer-Verlag, 1990.

[53] D. Seeley. A tour of the worm.
Available at http://world.std.com/ franl/worm.html#p2.

[54] L. Sherriff. Virus launches ddos for mobile phones.
Available at http:// www.theregister.co.uk/content/1/12394.html.

[55] R. Stone. Centertrack: An ip overlay network for tracking dos floods. *9th USENIX Security Symposium*, pages 199–212, August, 2000.

[56] Computer Emergency Response Team. Cert advisory CA-96.01: UDP port Denial-of-service attack, 1996.

[57] Computer Emergency Response Team. Cert advisory CA-97.28: IP Denial-of-service attacks, 1997.

[58] Computer Emergency Response Team. Cert advisory CA-99.17: Denial-of-service tools, 1999.

[59] Computer Emergency Response Team. Cert advisory CA-96.21: TCP SYN flooding and IP spoofing attacks, 24 September 1996.

[60] Computer Emergency Response Team. Cert advisory CA-96.26: Denial-of-service attack via ping, December, 1996.

[61] Computer Emergency Response Team. Cert advisory CA-98.01: Smurf IP Denial-of-service attacks, December, 1996.

[62] Computer Emergency Response Team. Cert advisory CA-95-01: IP Spoofing attacks and Hijacked terminal connections, January, 1995.

[63] A. Wagner, B. Plattner, and R. Hiestand. Experiences with worm propagation simulations. In *Proceedings of the First ACM Workshop on Rapid Malcode (WORM03)*, October 1999.

[64] C. T. Wang, C. H. Lin, and C. C. Chang. Signature schemes based on two hard problems simultaneously. In *the 17th International Conference on Advanced Information Networking and Applications*, pages 557–560, 2003.

[65] Ge' Weijers. re:client puzzle protocol.
Available at http://archives.neohapsis.com/archives/nfr-wizards/2000-q1/0558.html, 2000.

[66] M. Williams. Ebay, amazon, buy.com hit by attacks. IDG News Service, 9 February 2000.

[67] Y. Xiang, W. Zhou, and M. Chowdhury. A survey of active and passive defence mechanisms against ddos attacks. *Technical Report, TR C04/02, School of Information Technology, Deakin University, Australia*, March, 2004.

[68] B. Ziegler. Hacker tangles panix Web site. Wall Street Journal, 12 September 1996.