Analysis and mitigation of information exposure on interest management in massively multiplayer online games

Hao, Jianan

2017

Hao, J. (2017). Analysis and mitigation of information exposure on interest management in massively multiplayer online games. Doctoral thesis, Nanyang Technological University, Singapore.

<http://hdl.handle.net/10356/69436>

<https://doi.org/10.32657/10356/69436>

NANYANG TECHNOLOGICAL UNIVERSITY

Analysis and Mitigation of Information Exposure on Interest Management in Massively Multiplayer Online Games

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

by

Hao Jianan

Supervisor: Prof. Cai Wentong

School of Computer Engineering

2017

Abstract

Massively Multiplayer Online Games (MMOGs) have achieved unambiguously success over the last decade. The technology offers great chances for millions of geographically distributed people to connect together and enjoy immersed experience in the Virtual Environment (VE). Although most published games rely on client/server architecture today, the centralized architecture suffers from scalability issues. Alternatively, Peer-to-Peer (P2P) is considered as a challenging but reasonable solution to replace client/server architecture.

The network architecture is responsible for exchanging a number of game states among players in every round of the game. To make network efficient, Interest Management (IM) is commonly applied in games to reduce unnecessary network communications. However, without trusted server deployed by game operator, P2P based games usually rely on players' computers to perform IM. Since game programs can be eavesdropped or modified by players, IM on P2P architecture is vulnerable to Information Exposure (IE) attacks, i.e., the attack aiming for purposely obtaining the victim's sensitive information without authorization. IE attacks on P2P based IM lead to the expose of sensitive information such as player's position to the attacker. Mitigating such attacks is challenging. Firstly, P2P based IM is running on players' computers whose behaviors are unpredictable. Secondly, IE attacks are hardly to be detected because they can be performed without modifying the game program. Thirdly, the mitigation to IE attacks must avoid consuming excessive network bandwidth in order to fulfill the original purpose of IM, i.e. reducing unneces-

sary bandwidth consumption. Mitigation to IE attacks requires the above challenges to be solved but currently, solutions are still missing. This is the very task of this thesis, designing P2P based IM schemes that can mitigate IE attacks.

The thesis starts by formalizing IE attacks on IM schemes as interactive protocols with adversary behaviors. Criteria such as correctness, secrecy and bandwidth usage are proposed to verify candidate IM schemes' functionality and evaluate their mitigation effectiveness. Two novel IM schemes are proposed in this thesis to mitigate IE attacks. The first one is called SAFE which relies on privacy preserving protocol to obfuscate sensitive information. The second one is called PP2P which relies on an isolated execution environment called trusted block to protect sensitive information and operations. Especially, an implementation of trusted block is provided, which is based on hardware virtualization and trusted computing. Evaluations are applied to the proposed IM schemes via model checking and simulation, and the experiment results show both schemes have significant effects on mitigating IE attacks.

We hope this thesis can be valuable to academia especially on privacy preserving computation, hardware virtualization, trusted computing and formal verification. The proposed evaluation properties can become a widely used standard for evaluating IM schemes. We also expect our proposed schemes can be adopted by game industry to allow game users to enjoy fairer games, and to protect game operators from declined revenue caused by cheating issues.

Acknowledgments

I wish to render my heartfelt appreciation to my supervisor, **Prof. Cai Wentong**, for his continuous, remarkable, invaluable advice and guidance. Without his consistent support and help, the research would not have been so enriching and fulfilling.

I also wish to express my sincere thanks to **Dr. Zhou Suiping**, who has offered me this precious opportunity to be a Ph.D. candidate in Nanyang Technological University.

My appreciation also goes to the technician, **Mdm. Ng-Goh Siew Lai Irene** at Parallel and Distributed Computing Centre for her expert support, my fellow colleagues for their generous sharing of ideas and School of Computer Engineering for offering me this great research environment.

Last but not least, I would like to thank my wife, **Xu Xiang**, for her continuous support on both research and life; my parents and friends for their care and love, from which I have gained confidence and support to finish this adventure.

Hao Jianan

2017

Contents

Abstract	i
Acknowledgments	iii
List of Figures	x
List of Tables	xii
List of Notations	xiv
Glossary	xv
1 Introduction	1
1.1 Background and Motivation	1
1.1.1 C/S and P2P Architectures	2
1.1.2 Interest Management	3
1.1.3 Information Exposure Attacks on IM Scheme	5
1.2 Objectives and Contributions	7
1.3 Organization of the Thesis	8
2 Related Work	9
2.1 Client/Server and Peer-to-Peer Architectures	9
2.1.1 C/S Architecture	10
2.1.2 P2P Architecture	10
2.1.3 P2P Challenges	11
2.2 Interest Management	12

2.2.1	IM Classification	12
2.2.2	Implementation of IM Schemes in Existing P2P DVEs	14
2.2.3	IM Evaluation	20
2.3	Cheat and Mitigation	22
2.3.1	Existing Cheats	22
2.3.2	Cheating Classification	25
2.3.3	Mitigation on Cheats	27
2.4	Summary	31
3	Problem Statement	33
3.1	Definition of IM	33
3.1.1	Functionality of IM	33
3.1.2	Definition of IM Schemes	34
3.1.3	Notations and Assumptions	35
3.1.4	A Lite Game	37
3.2	Modeling CS and SP2P Schemes	38
3.2.1	CS	38
3.2.2	SP2P	39
3.3	Definition of IE Attacks	41
3.3.1	Adversary Model	41
3.3.2	IE Attacks	43
3.4	Properties of IM Schemes	43
3.4.1	Correctness	43
3.4.2	Secrecy-1	44
3.4.3	Secrecy-2	44
3.4.4	Bandwidth Usage	45
3.5	Summary	46

4	SAFE	47
4.1	Design Rationale	47
4.1.1	Modeling f_{sl} as Set Membership Testing	48
4.1.2	Detection Region	49
4.2	Privacy Preserving Protocol	50
4.2.1	KM-1 Protocol	51
4.2.2	Detection of Malicious Behavior	52
4.2.3	Supplementary Protocol	53
4.3	SAFE Protocol	58
4.4	Summary	59
5	PP2P	61
5.1	Trusted Block	62
5.1.1	Design Rationale	63
5.1.2	Implementation	66
5.2	PP2P Protocol	74
5.2.1	Trusted IM	75
5.2.2	Legacy IM	78
5.3	Summary	79
6	Formal Verification	81
6.1	System Modeling	82
6.1.1	Model Checking and CSP# Language	82
6.1.2	Modeling the Lite Game	85
6.1.3	Abstraction of IM-related States	86
6.1.4	Behavior of IM Schemes	89
6.1.5	Adversary Behavior	92

6.2	Property Specifications	95
6.2.1	Correctness	95
6.2.2	Secrecy-1	95
6.3	Modeling IM Schemes	97
6.3.1	CS	98
6.3.2	SP2P	98
6.3.3	SAFE	100
6.3.4	PP2P	102
6.4	Experiment Results and Discussions	105
6.4.1	Correctness	106
6.4.2	Secrecy-1	106
6.5	Summary	107
7	Quantitative Evaluations	109
7.1	Game Simulation	110
7.1.1	Simulating IM-related States	110
7.1.2	Mobility Model	111
7.2	Property Evaluation	112
7.2.1	Secrecy-2	112
7.2.2	Bandwidth Usage	115
7.3	Modeling IM Schemes	115
7.3.1	CS	115
7.3.2	SP2P	116
7.3.3	SAFE	118
7.3.4	PP2P	119
7.4	Experiment Results and Discussions	121

7.4.1	Simulation Parameters	121
7.4.2	Secrecy-2	122
7.4.3	Bandwidth Usage	123
7.4.4	Discussions	124
7.5	Summary	124
8	Conclusions and Future Work	127
8.1	Conclusions	127
8.2	Future Work	129
8.2.1	Improvement of Network Performance	129
8.2.2	Lightweight Implementation of Trusted Block	129
8.2.3	More Realistic Simulation	130
A	Background	131
A.1	Privacy Preserving Protocol	131
A.1.1	Public Key Cryptography	131
A.1.2	Cryptographic Hash Function	133
A.1.3	Public Key Infrastructure	134
A.1.4	Homomorphic Encryption	135
A.2	Hardware Virtualization	135
A.2.1	Processor Virtualization	136
A.2.2	Memory Virtualization	137
A.2.3	Security Enhancement Based on Virtualization	138
A.3	Trusted Computing	139
A.3.1	Trusted Platform Module	139
A.3.2	Measurement	140
A.3.3	Remote Attestation	141

A.3.4	Key Generation and Distribution	142
A.3.5	Known Issues	143
A.4	Summary	144
B	Source Code of Model Checking	145
B.1	im.csp	145
B.2	cs.csp	148
B.3	sp2p.csp	150
B.4	safe.csp	153
B.5	pp2p.csp	156
	Author's Publications	161
	References	162

List of Figures

3.1	The behaviors of Alice and Bob in the lite game	37
3.2	Protocol of $\mathcal{P}_{cs}$	38
3.3	Protocol of $\mathcal{P}_{sp2p}$	40
4.1	Algorithm of KM-1	51
4.2	Protocol of $\mathcal{SP}_{sub}$	54
4.3	Protocol of $\mathcal{SP}_{pub}$	56
4.4	Protocol of $\mathcal{P}_{safe}$	57
4.5	Sequence Diagram of SAFE	58
5.1	The trusted block and the legacy system	62
5.2	Secure communication	64
5.3	Implementation architecture	67
5.4	Sequence diagram of public function invocation	68
5.5	Sequence diagram of key generation, verification and distribution	73
5.6	The publisher's trusted IM (<i>pub_trust</i>)	76
5.7	The subscriber's trusted IM (<i>sub_trust</i>)	77
5.8	Legacy IM (<i>pub_legacy</i> and <i>sub_legacy</i>)	78
6.1	Model Checking	82
6.2	Modeling the lite game	85
6.3	Modeling the nondeterministic state of <i>aoiiner</i>	89

6.4	Modeling the nondeterministic state of <i>insl</i>	89
6.5	An example of modeling benign subscriber in $\mathcal{P}_{sp2p}$	90
6.6	An example of modeling HBC behavior	92
6.7	An example of modeling the subscriber's malicious behavior on SL update	93
6.8	Modeling overridden results of relationship functions	93
6.9	Modeling of publisher's evaluation of <i>insl</i>	93
6.10	Correctness property	96
6.11	Secrecy-1 property	96
6.12	Modeling of protocol $\mathcal{P}_{cs}$	97
6.13	Modeling of protocol $\mathcal{P}_{sp2p}$	99
6.14	Modeling of protocol $\mathcal{P}_{safe}$	101
6.15	Modeling of protocol $\mathcal{P}_{pp2p}$	103
6.16	Modeling of subscriber's trusted block (<i>sub_trust</i>) in protocol $\mathcal{P}_{pp2p}$. . .	104
6.17	The counter example leading $\mathcal{P}_{sp2p}$ to <i>NegSecrecy1</i>	106
6.18	The counter example leading $\mathcal{P}_{safe}$ to <i>NegSecrecy1</i>	107
7.1	The visualization of pseudo peers (SAFE)	113
7.2	Secrecy-2 of IM Schemes (the higher the better)	122
A.1	Public key cryptography applications	132
A.2	Cryptographic hash function	133
A.3	Processor virtualization	136
A.4	Memory virtualization	137
A.5	Trusted platform module	140

List of Tables

2.1	Summary of existing IM schemes	15
3.1	General notations for modeling IM schemes	35
4.1	Summary of candidate privacy preserving protocols on privates set operations	50
5.1	List of related TPM commands	69
6.1	Symbolic representations of IM-related states	87
6.2	Relationship functions	87
6.3	Verification results of correctness and secrecy-1	105
7.1	Definitions of IM-related states and functions in simulation	110
7.2	Refinement operations	114
7.3	Typical lengths of basic elements	115
7.4	Information learned in protocol $\mathcal{P}_{cs}$	115
7.5	Information learned in protocol $\mathcal{P}_{sp2p}$	116
7.6	Information learned in protocol $\mathcal{P}_{safe}$	118
7.7	The lengths of major messages in protocol $\mathcal{P}_{safe}$	119
7.8	Information learned in protocol $\mathcal{P}_{pp2p}$	119
7.9	The lengths of major messages in protocol $\mathcal{P}_{pp2p}$	121
7.10	Typical values for the simulation parameters	121
7.11	Bandwidth usage of IM schemes	123

7.12 Properties of IM Schemes	124
---	-----

List of Notations

V	The set of all reachable positions in the virtual environment
pos_x	Position of player x
aoi_x	AOI of player x
er_x	ER of player x
dr_x	DR of player x
ra_{aoi}	Radius of AOI
ra_{er}	Radius of ER
ra_{dr}	Radius of DR
f_{im}	Function of interest management
f_{sl}	Function of SL update
f_{rel}	Function to filter relevant game state
$E_k(x)$	Encryption of x by key k
$D_k(x)$	Decryption of x by key k
$H(x)$	Cryptographic hash value of x
l_{crypt}	Length of $E_k(x)$ or $D_k(x)$ output
l_{hash}	Length of $H(x)$ output
l_{er}	Length of ER update
l_{gs}	Length of game state update
$\ x, y \parallel$	Euclidean distance between x and y
$ s $	Cardinality of set s
$x \parallel y$	Concatenation of x and y

Glossary

AIK	Attestation Identity Key
AOI	Area-of-Interest
C/S	Client/Server
CSP	Communicating Sequential Processes
DR	Detection Region
DVE	Distributed Virtual Environment
EK	Endorsement Key
ER	Extension Region
FPS	First Person Shoot
HBC	Honest-But-Curious
IE	Information Exposure
IM	Interest Management
KM-1	A privacy preserving protocol of disjointness test
LTS	Labeled Transition System
MMOG	Massively Multiplayer Online Game
NPC	Non-Player Character
OS	Operating System
P2P	Peer-to-Peer
PAT	Process Analysis Toolkit
PCR	Platform Configuration Register
PMF	Probabilistic Mass Function
PP2P	Protected P2P Scheme
RPG	Role-Playing Game
RSA	A public-key cryptosystem proposed by Rivest, Shamir and Adlema
RTS	Real-Time Strategy
RTT	Round-Trip Time
SAFE	Secure Area of interests FiltEr Scheme
SGX	Software Guard Extensions
SL	Subscriber List
SP2P	Standard P2P Scheme
SRK	Storage Root Key
TC	Trusted Computing
TPM	Trusted Platform Module

TTP	Trusted Third Party
VE	Virtual Environment
VM	Virtual Machine

Chapter 1

Introduction

This chapter provides background and motivation of this research, including its objectives and contributions. The organization of this thesis is also given.

1.1 Background and Motivation

With the development of computer graphics and Internet, Massively Multiplayer Online Games (MMOGs) become popular to connect geographically distributed people playing in the Virtual Environment (VE). Without the need of sitting together, millions of MMOG players can start their gaming journeys in front of their computers for excited experiences. Based on game genres such as First-Person Shooter (FPS), Role-Playing Game (RPG) and Real-Time Strategy (RTS), each player controls an entity, a.k.a., avatar, in the virtual environment to frag¹ enemies, spell magic, explore unknown virtual world, upgrade weapons and hunt monsters. MMOGs are very large distributed systems connecting thousands of computers. The underlying network architecture is the key to support communications among these computers. In general, Client/Server (C/S) and Peer-to-Peer (P2P) are two common architectures adopted by MMOGs.

¹A term refers to ‘kill an enemy by a grenade’, which is popularized in video games since Doom, the first FPS game in the world.

1.1.1 C/S and P2P Architectures

In most cases, an MMOG is built based on C/S architecture because of its maturity. The architecture involves a server maintained by game operator to host a game session to which multiple clients connect. Clients are usually computers controlled by human players. Clients receive commands from game controllers, convert the commands into actions, and transmit the actions to the server. The server maintains the states of all the players and objects in the VE. Upon actions are received, the server simulates them according to game rules and performs necessary changes on the states of involved players and objects. The server is also responsible for sending the latest states to the clients who are interested. Clients can thus render these latest states for human players to react.

While C/S architecture has succeeded in existing MMOGs, it has several known drawbacks [11, 81, 82, 99, 134]. Firstly, a server responsible for simulating all game commands can be easily overloaded in the sense of computational capability and network bandwidth under increasing number of clients. Secondly, when players interact with each other, C/S architecture forces their clients to communicate with central server regardless of network topology. When clients connect to the server with a high latency network, significant delay is observed by users. Thirdly, a server suffers from a single point of failure, which may affect the service availability. Lastly, maintaining a server that can support a large number of growing subscribers is costly. It could prevent start-up game operators from publishing their creative game work.

P2P architecture is considered as a reasonable solution to the above issues. Compared to the C/S architecture, P2P can offload heavy computations and network activities to peers, thus providing excellent scalability, optimal network latency and low operating cost. In particular, players (peers) connect each other directly in a P2P game. Every peer maintains his/her game states and contributes computing and storage resources to the game. Therefore, the number of peers grows adaptively with increasing game

subscribers, which provides scalability. The decentralized design also saves investments on server maintenance. Nevertheless, P2P architecture also has disadvantages such as vulnerability of cheating issues, difficulty in maintaining state consistency and overhead of tolerance of disconnected peers. These drawbacks require more sophisticated design to overcome.

In recent years, a number of research work has been done in P2P based MMOGs [27, 33, 48, 49, 55, 81, 99, 133, 134, 136, 140]. It covers various areas such as interest management, state consistency, distributed state persistence, etc. Among these areas, interest management is particularly an active one.

1.1.2 Interest Management

Interest Management (IM) is a vital component to manage player's interests, for the purpose of reducing unnecessary communication between server/client or among peers. In a game, various events can occur at any given time, which results in a number of game state updates. Players must continuously react according to the latest game states of other players and objects in the virtual environment. To this end, game state update of a player must be acknowledged by some others. However, publishing these state updates to every player is technically infeasible and unnecessary. In fact, the game usually adopts certain rules that can confine particular player's *interest*. For example, in an FPS game, each player has limited visibility range. Any change outside this range needs not to be rendered. Thereby knowing game states of those out-of-range players and objects is unnecessary. In this case, receiving only state updates on those interested players and objects is sufficient for a player to play the game, which is the very task for IM.

IM introduces Area-of-Interest (AOI) and Subscriber List (SL) as two major structures for reducing unnecessary state updates. AOI defines the region in which a player is interested. Any game state updates of other players and objects residing in this region

should be received by the owner of the AOI. By definition, the player who owns AOI and receives game states is called state *subscriber*. The other player who provides game state updates is called state *publisher*. In general, the publisher maintains an SL which contains all subscribers whose AOIs cover the publisher's position. Every time when the publisher's game state is updated, the subscriber will obtain this update if he/she is in the SL or in other words, if the publisher's position is contained in the subscriber's AOI. As players move and perform other game actions over time, the publisher's SL and the subscriber's AOI are dynamically changed.

C/S and P2P based architectures have very different ways to maintain AOI and SL. In C/S architecture, maintaining AOI and SL is completely handled by the central server. Clients only receive interested game state updates. That is to say, the server is responsible for keeping the information of the subscriber's AOI and the publisher's position. Thus, there is no explicit messages passing between the subscriber and the publisher. Both AOI and SL updates can be fully handled by the central server, and game state updates are sent only when necessary.

In P2P architecture, AOI and SL updates are performed by peers in a fully distributed manner. The publisher's game state/SL and the subscriber's game state/AOI are separately maintained by themselves. However, the publisher needs the subscriber's latest AOI to compare with the publisher's latest position in order to update its SL. This introduces additional network traffic for AOI updates. As AOI may change frequently during the game, sending the subscriber's every AOI update to the publisher may consume significant network bandwidth. This becomes a problem as the original purpose of IM is to reduce unnecessary network communication.

To address network problem, a mechanism called area-based filtering [17] is widely used. It optimizes the network communication by introducing Extended Region (ER)²

²To not be confused, ER defined in this thesis is not for providing better state consistency.

which is an approximation to AOI. ER is maintained as a region that fully covers the AOI. As ER can be kept unchanged as long as it covers the AOI, it is updated less frequently than the AOI. So, to reduce network traffic, the subscriber can send ER update, instead of AOI update, to the publisher. Game state updates received by the subscriber are from the publisher whose position is inside the subscriber's ER. It is possible that the publisher's position is not inside the subscriber's AOI even if the subscriber receives the publisher's game state update. Such a game state update is called an irrelevant update and should be discarded by the subscriber. This area-based filtering scheme is a typical and practical P2P IM scheme to solve the problem of network bandwidth consumption. To be clear, we name this IM scheme as Standard P2P (SP2P). We will use it throughout the thesis for comparison with our proposed P2P based schemes.

1.1.3 Information Exposure Attacks on IM Scheme

Besides network performance, IM also suffers from cheating issues which are common challenges in designing P2P games. One of these cheats is called Information Exposure (IE) attack [133]. IE attack refers to the behavior of purposely obtaining the victim's sensitive information without authorization. The attack gives the attacker additional but unauthorized information of the victim. Such information grants the attacker advantages in the competition. The violation of game balance harms the victim's user experience. If it is not fixed by the game operator, the IE attack usually results in the victim quitting game subscription.

IM is severely affected by IE attacks because IM is frequently performed in a game containing sensitive information such as one's position or other states related to the position. C/S based IM by nature prevents IE attacks because the IM scheme is executed by the central server. The game operator has full control over the server and guarantees its functionality. Clients controlled by players only receive game states from the server and thereby cannot affect the execution of the IM scheme.

On the contrary, P2P based IM is vulnerable to IE attacks. Both functionality and security of IM schemes rely on whether they are faithfully carried out. For a P2P IM scheme, its protocol is collaboratively performed on each peer's computer. Since the computer is controlled by its owner, i.e., a player competing in the game, it is possible that the program running the protocol is eavesdropped or compromised. This will lead to an IE attack, not to mention that the player also has motivation to perform the IE attack for unauthorized information of other benign players. The design of SP2P does not mitigate IE attacks. For example, the subscriber's ER is not supposed to be known by the publisher. However, the publisher as an attacker can obtain this information when the ER update is received. The attacker can utilize this information to guess where the subscriber is.

Mitigating IE attacks on P2P based IM is challenging in several ways. Firstly, P2P based IM is running on players' computers whose behaviors are unpredictable. Unlike the central server maintained by game operator, a P2P IM scheme is performed on the computers that are controlled by players. In the scenario of MMOG, these computers are open platforms and their owners, a.k.a., players, are able to decide what can be run on it (including the program to perform IE attacks). Secondly, IE attacks are hardly to be detected. Unlike attacks aiming to modify states, IE attacks can be performed without changing the behavior of IM. Considering the above example of IE attack in SP2P, from the view of the subscriber, the publisher (attacker) has identical behavior to what a benign one has. It makes the attack hardly detectable by a remote party. Thirdly, the mitigation to IE attacks must keep network communication as low as possible. It is because the original purpose of IM is to reduce unnecessary communication. Mitigation solutions must fulfill this purpose by not introducing excessive network load. In order to mitigate IE attacks, the above challenges must be addressed. Unfortunately, mitigation solutions of IE attacks on P2P IM are still missing. We argue that IE attacks on P2P

based IM are critical issues that must be solved. To pave the way to practical P2P games, this thesis will analyze IE attacks on P2P IM and propose efficient and practical mitigation solutions.

1.2 Objectives and Contributions

The objective of this thesis is to design P2P based IM schemes that can mitigate IE attacks. The mitigation effectiveness of these proposed schemes is validated by solid evaluation approaches. In sum, this thesis has the following contributions.

- *Analysis and evaluation criteria*: we formalize the IE attacks on IM schemes as a study of interactive protocols. The general functionality of IM is formally defined and each IM scheme is modeled as an interactive protocol. We propose correctness, secrecy (including secrecy-1 and secrecy-2) and bandwidth usage as the most essential properties to evaluate an IM scheme. In particular, secrecy-2 is a uniform indicator to quantitatively measure the effect of mitigation to IE attacks.
- *SAFE*: Secure Area of interests FiltEr (SAFE) is an IM scheme that relies on privacy preserving computation to obfuscate sensitive information. As a cryptography-based solution, SAFE can be deployed on most existing platforms. Especially, SAFE is able to detect malicious attacking behavior with the help of Trusted Third Party (TTP).
- *PP2P*: Protected P2P (PP2P) is another IM scheme proposed in this thesis. The PP2P designs a protected environment called trusted block. All sensitive information involved in the IM can be stored and processed in the trusted block. A possible implementation solution based on virtualization and trusted computing is provided.

- *Formal verification*: the formal verification is applied to verify functional properties such as correctness and secrecy-1. Each scheme is modeled by CSP# language and automatically tested. For unsatisfied properties, counter examples will be given.
- *Quantitative evaluation*: simulation based approach is employed to evaluate properties such as secrecy-2 and bandwidth usage. Especially, Monte Carlo simulation is used in the secrecy-2 evaluation. Based on experiment results, practical considerations are provided for choosing a P2P based IM scheme.

1.3 Organization of the Thesis

The rest of this thesis is organized as follows. Chapter 2 reviews the existing work related to this research. Chapter 3 defines the problem of this thesis. Chapter 4 describes SAFE, a P2P scheme based on privacy preserving computation to mitigate IE attacks. Chapter 5 describes PP2P, another P2P scheme based on trusted block to mitigate IE attacks. Chapter 6 formally verifies the correctness and secrecy-1 properties. Chapter 7 quantitatively evaluates the secrecy-2 and bandwidth usage properties. Chapter 8 concludes the thesis and discusses future work.

Chapter 2

Related Work

In this chapter, we review the existing work related to the IM and cheating issues in MMOG. Section 2.1 covers 1) strength and weakness of C/S and P2P architectures, and 2) key challenges that P2P games are facing. The rest of this chapter focuses on 1) interest management schemes, and 2) cheating issues and their mitigation methods.

2.1 Client/Server and Peer-to-Peer Architectures

The MMOG can be built based on different architectures such as C/S and P2P. Most network games use C/S architecture, where a central and trusted server connects all players together and runs the game. This game technology has proven to be a commercial success and many MMOGs are based on it [97], such as World of Warcraft (WoW) [18]. However, the explosive growth of game subscribers and the limitation of broadband connection for home players has resulted in a bottleneck presented in the centralized C/S architecture [20, 133]. An alternative to this architecture is decentralized P2P architecture. In recent years, P2P network have become an active research topic [23, 29, 45, 50, 68, 75, 77, 81, 123]. Its application on MMOG has also attracted high attention [14, 25, 28, 43, 47, 109, 115, 134, 135, 140]. We review two architectures respectively in the rest of this section.

2.1.1 C/S Architecture

In C/S architecture, central server handles all tasks such as event simulation, interest management, etc. The state of the Virtual Environment (VE) is maintained by the server. Clients controlled by individual players connect to the central server via Internet. They are responsible for sending commands and receiving game states for rendering. With rendered pictures on the screen, the player controls his/her game avatar for response. The operation commands are sent from the client to the central server which updates states according to the events raised by the commands. After that, the server publishes this game state updates to the clients who are interested. Those clients can thus render the change of VE to their screens according to the state updates for the players to react. MASSIVE-3 [56] and Community Place [84] are two typical frameworks using the C/S architecture.

Although C/S architecture guarantees players can only receive authorized information of others, the centralized design has many drawbacks [11, 81, 82, 99, 134] including: 1) the scalability is poor as the single server is limited to network capability and computing resources; 2) the latency is not optimal; 3) the server is a single point of failure which may affect the service availability; and 4) server maintenance is costly.

A variation of C/S architecture is the multi-server architecture. It upgrades the single server into multiple ones which connect to each other via high-speed internal network. This change eliminates the single point of failure and offers better latency. However, the scalability is still limited as the servers are responsible for handling all critical and heavy tasks. To our knowledge, Everquest [41] is built based on multi-server architecture.

2.1.2 P2P Architecture

The P2P architecture is an alternative to solve the above issues. In this architecture, the central server no longer exists (or is only responsible for light tasks, such as user

authentication and the arbitration of conflicted scenarios). The client, a.k.a. peer, is responsible for maintaining partial states of the VE, which scales to the increasing number of players. Without critical computation on centralized server, P2P architecture naturally eliminates the single point of failure. In addition, as the interaction is performed directly from a peer to another, the latency can be optimized. MiMaze [49] is the first fully distributed architecture for online games. However, it relies on IP multicast to give every peer a complete replication of the VE. Obviously, this approach does not scale when the number of peers increases. Knutsson *et al.* [81] propose a P2P framework called SimMud for the MMOG based on the structured overlay Pastry [112], which improves the scalability. DIVE [27] and Solipsis [45] are the other two game frameworks using the P2P architecture.

2.1.3 P2P Challenges

Compared to the mature C/S architecture, currently, P2P architecture still faces many challenges which prevent it from being widely used in practice. Some researchers investigate fundamental challenges of peer-to-peer gaming. Neumann *et al.* [99] suggest several problems that remain imperfectly solved for P2P, such as distributed storage, game state consistency and management, network latency and cheating issues. Fan *et al.* [42] propose six key challenges of P2P systems: interest management, game event dissemination, Non-Player Character (NPC) host allocation, game state persistence, cheating mitigation and incentive mechanism.

Some other researchers pay special attention to certain challenges in P2P MMOGs and try to give a guide on how to solve them. Miller and Crowcraft [95] argue that network bandwidth is a must-do challenge in P2P design. In their paper, they simulate the MMOG game WoW using typical broadband speed. Based on the experimental result, they conclude P2P related technologies and protocols are not deployable given typical residential broadband infrastructure.

Gilmore and Engelbrecht [52] offer a survey to explain state persistency as one of the key challenges in P2P MMOGs. In the survey, the persistency techniques are proposed using super peer storage, overlay storage, hybrid storage, and distance-based storage. Key characteristics are defined to evaluate each state persistency technique, such as scalability, fairness, reliability, responsiveness, and security.

2.2 Interest Management

Interest management (IM) is a filtering mechanism that allows a peer to receive only a subset of information that is relevant to him/her [25, 98]. By specifying what data should be sent and what should be received, the numbers of transmitted messages are reduced.

2.2.1 IM Classification

IM for Distributed Virtual Environments (DVEs) has been studied in different areas, such as military and academic simulation systems [98], social media and commercial applications including MMOGs [28, 87]. A number of IM schemes have been proposed thereafter. Some work has been published to classify them. Note that, same terminology in different classifications may refer to different semantics.

Liu and Theodoropoulos [87] classify the existing IM schemes into six major categories, namely zone-based, aura-based, class-based, visibility-based, hybrid, and content streaming-based. Zone-based and aura-based categories are classified according to how the AOI is represented. Zone-based schemes (a.k.a., cell-based and grid-based) divide the VE into subdivisions (i.e., zones) and limit a peer's interaction into a number of zones. Aura-based (a.k.a., area-based) schemes filter data from an area of circle or eclipse centered at the object. Visibility-based and class-based are classified according to how the shape and size of AOI is defined. Visibility-based schemes filter data based on the object's visibility. Class-based schemes filter data according to whether or not candidate

object satisfies certain requirements. For example, in a military simulation scenario, radar should detect all nearby aircrafts except stealth fighters. Therefore, the radar in the simulation should not receive updates from the stealth fighters. Hybrid schemes employ two of above schemes for filtering. Content streaming is a technique to update static data (e.g., terrain information) online.

Cater *et al.* [28] discuss AOI management specifically related to P2P MMOG. In the paper, they classify AOI management as spatial-based, geographic-based and hybrid-based schemes. Spatial-based schemes define AOI by the properties of peers, which is comparable to aura-based category in [87]. Auro-based or Voronoi-based [131] schemes belong to spatial-based schemes. Geographic-based schemes define AOI by dividing VE into small zone, which is comparable to zone-based category in [87]. Examples of geographic-based AOI management are tile-based or hexagonal geographic schemes [68], hierarchical schemes [3] and triangular geographic schemes [19]. Hybrid schemes utilize aspects of both spatial and geographic information.

Morse [98] classifies IM schemes according to communication model, filtering focus and IM Domain of Responsibility (DOR). Communication model refers to the model for message transmission. Unicast, broadcast, and multicast are three types of communication models. Filtering focus refers to whether data filtering is based on intrinsic or extrinsic characteristics of objects. The intrinsic characteristics of an object are the values of its own attributes. The extrinsic characteristics of an object are the factors outside it, such as environmental information like locations. An IM DOR is the area in multi-dimensional space in which the IM is responsible for managing data transmission. DORs can be either static or dynamic.

Carlini *et al.* [25] describe a classification of IM schemes, with regard to the following four categories: architecture, P2P overlay, AOI precision and AOI shape. A further classification of the architecture can be performed by considering the difference in roles

the nodes play in the infrastructure. The nodes could have different functionalities (hierarchical architecture); or on the contrary, all nodes function the same in the network (flat architecture). The P2P overlay can be either structured (such as Distributed Hash Table (DHT) [9, 108, 112, 141]) or unstructured (such as gossip-based [73] or Voronoi networks). AOI precision can be discrete or continuous. The discrete AOI precision is used in those approaches that perform a static, grid-like partitioning of the virtual environment, which is comparable to zone-based category in [87]. The continuous AOI precision is dynamically computed as entities move, which is comparable to aura-based category in [87]. AOI shape can be circle, square, rectangle, or hexagon.

Among the above mentioned work, we think [25] is more systematic. Therefore, we choose it to categorize IM schemes surveyed in the next sub-section.

2.2.2 Implementation of IM Schemes in Existing P2P DVEs

In traditional network, IM is achieved either through message filtering from sender side, or by using IP multicast to send data to a group of interested receivers. While the implementation of IM is straightforward in the C/S architecture, it is however challenging in the P2P architecture [87, 140]. Firstly, managing data from distributed peers is difficult. Secondly, such a system is more vulnerable to attacks (e.g., IE). Some existing IM schemes for P2P DVEs will be elaborated in the following paragraphs. Table 2.1 summarizes their properties.

SimMud

SimMud [81] is one of the most popular examples to build DVE on structured P2P architecture based on DHT. The key to the feasibility of SimMud (or other P2P game architecture) is locality of interest [98]. In SimMud, the virtual world is divided into disjoint zones. Each player can stay in only one zone at a time, and receive all updates from the zone. Each zone is hashed and assigned to a node. All nodes form a Pastry

Table 2.1: Summary of existing IM schemes

IM Scheme	Architecture	Overlay	AOI
SimMud	Hierarchical	Structured (DHT)	Rectangular (discrete)
Colyseus	Flat	Structured (DHT)	Rectangular (continuous)
DonnyBrook	Flat	Unstructured	Unspecified (continuous)
MOPAR	Hierarchical	Mixed	Hexagonal (continuous)
VON	Flat	Unstructured (Voronoi)	Circular (continuous)
pSense	Flat	Structured (Gossip)	Circular (continuous)
Hybrid IM scheme	Flat	Unstructured	Unspecified (discrete)

overlay [112]. Each node thus becomes a coordinator of the corresponding zone, which is responsible for 1) synchronizing all updates to shared entities in the zone; and 2) disseminating updates to players of the zone using a Pastry-based Application Level Multicast (ALM) called Scribe [29].

According to classifications in [25], SimMud uses hierarchical architecture. Its P2P overlay is structured (based on DHT) and the AOI is discrete and rectangular.

Colyseus

Another influential example is *Colyseus* [17]. Colyseus is built as a DVE based on Mercury [15]. It uses both traditional randomized DHT and a range-queriable DHT to distribute objects to peers and coordinate object updates within the AOI of a peer. Each object has one primary copy assigned to a peer and may have several replicas residing on different peers to enable quick local access to the object. The primary owner of the object is responsible for synchronizing the updates of the object. However, it is not affordable for the owner to send AOI updates in each frame as this causes too much communication overhead. Instead, it periodically publishes the updates of the object to the peers who are interested. This introduces a new problem: not sending updates every frame might cause inconsistency. For example, a short-lived object (e.g., missile) might not be consistently captured by the IM scheme. To address this issue, Colyseus employs proactive replication. It allows the creator of the missile to actively replicate the states of

the missile to the ones who are interested in the creator. This would increase unnecessary communication because not every state receiver is interested to all newly created objects.

According to classifications in [25], Colyseus uses flat architecture. Its P2P overlay is structured (based on range-queriable DHT) and the AOI is continuous and rectangular.

DonnyBrook

Bharambe *et al.* [16] adopt the concept of multiresolution filtering in their aura-based IM scheme DonnyBrook. In DonnyBrook, players receive high-fidelity updates (e.g., updates in every frame) from the entities in their interest set, while receive updates in low fidelity (e.g., updates rendered less frequently in frames) from everyone else. The interest set is obtained by evaluating values of attention, which is calculated based on three aspects: 1) proximity: players are more likely to pay attention to objects nearby; 2) aim: players are also more likely to pay attention to specific orientation where there are objects they are aiming at; and 3) interaction recency: players who recently interacted are more likely to pay attention to each other. Like most of other IM schemes, DonnyBrook has tradeoff. Firstly, extra computational overhead is introduced to determine the interest set of the participants. Secondly, sending low-fidelity updates of entities outside interest set might risk missing important events that are supposed to be captured by IM, and thus resulting in incorrect simulation. Therefore, this approach is only desirable to DVEs with limited bandwidth and/or intensive data communication.

According to classifications in [25], DonnyBrook uses flat architecture. Its P2P overlay is unstructured and the AOI is continuous. The AOI shape is unspecific.

MOPAR

Yu and Vuong [140] present a fully distributed P2P infrastructure to tackle interest management issues in MMOGs. Their architecture is a hybrid IM scheme, which takes the advantage of both structured P2P overlay (such as DHT) and unstructured P2P architecture. The system divides the virtual world into hexagonal cells. Three types of

nodes are introduced in MOPAR: master nodes, slave nodes, and home nodes. Home nodes are virtual nodes and they are responsible for master/slave nodes registration and deregistration. Each cell has one home node and this correspondence is kept unless the overlay has joining/leaving events. Master nodes are responsible for the distribution of data. They play the main role for IM. Each cell has only master node and the correspondence is dynamically changed when nodes move across the cells. Slave nodes in each zone receive updates from master nodes. A newly joined node first queries the home node (DHT query), and either registers as a master node if there is no master node registered for this zone, or becomes a slave node. That is, at the inter-cell level, P2P overlay is structured; however at the intra-cell level, the overlay is unstructured.

In the original protocol proposed by Yu *et al.*, the number of master nodes in each cell is fixed to (at most) one. This causes heavy load if many players concentrate in a specific cell. Fujita [46] proposes an improved version of MOPAR. The basic idea is to dynamically adjust the number of master nodes in each cell using a threshold-based merge-and-split mechanism. The improved method reduces the response time of MOPAR by about 50%.

According to classifications in [25], MOPAR uses hierarchy architecture (master nodes, slave nodes, and home nodes). Its P2P overlay is mixed-structured (based on DHT) at the inter-cell level and unstructured at the intra-cell level. The AOI is continuous and hexagonal.

VON

Hu *et al.* [67] propose a scalable IM scheme called VON. VON uses Voronoi diagram to partition the VE into a number of Voronoi regions, where each region contains a peer. For a given peer, AOI neighbors are the peers whose positions are within its AOI. Enclosing neighbors are those whose regions are next to the given peer. Among AOI neighbors, boundary neighbors are those whose enclosing neighbors are partially outside

the AOI. The neighbors are dynamically changed. When a peer moves, its position updates are sent to both AOI neighbors and enclosing neighbors, in order to maintain Voronoi diagram and update each peer's neighbor lists. VON achieves scalability by dynamically constructing Voronoi diagram.

According to classifications in [25], VON uses flat architecture. Its P2P overlay is unstructured and is based on Voronoi diagram. The AOI is continuous and circular.

pSense

Schmieg *et al.* [115] propose a P2P solution to handle IM and support dynamic behavior of MMOGs. The IM scheme is an aura-based filtering, called *pSense*. Every node (denoted as local node) in a pSense network maintains two lists: near node list and sensor node list. The near node list contains the nodes (a.k.a., near nodes) within the AOI of the local node. The sensor node list contains the nodes (a.k.a., sensor nodes) outside but close to the AOI of the local node. The major responsibility of sensor nodes is to introduce newly discovered nodes to the local node, in order to maintain two lists. The idea of pSense is similar to VON [67] as sensor nodes in pSense are comparable to boundary neighbors in VON.

According to classifications in [25], pSense uses flat architecture. Its P2P overlay is unstructured and is based on gossip. The AOI is continuous and circular.

Hybrid IM scheme

Pan *et al.* [102] propose a hybrid IM scheme that utilizes cell-based IM [36] to reduce the communication overhead (i.e., AOI updates) in area-based IM [17]. Traditionally, the AOI update is a costly overhead. As an object's AOI changes dynamically with regard to its various positions, the AOI updates are sent to all other entities without considering their relevance. By utilizing the cell-based IM, an entity sends its AOI updates only to those entities close to its current location. Therefore, the number of AOI updates is expected to be greatly reduced. Their scheme is simulated and evaluated in a MMOG scenario to prove its effectiveness.

According to classifications in [25], the hybrid IM scheme uses flat architecture. Its P2P overlay is unstructured. The AOI is discrete and can be any shape.

A³

Instead of circular shape, the Field of View (FOV), a fan-shaped area that defines the visual field of a player, is often used in DVEs for AOI specification. However, this approach has a problem that when a player turns around very quickly at a large angle (e.g., 180 degree), a lag effect might occur to the player's sight. This happens because the player does not receive all needed data due to the limited bandwidth and network latency. Bezerra *et al.* [14] discuss this problem of simply using FOV instead of circular AOI, and propose a hybrid AOI called A^3 AOI by combining of both circular AOI and FOV. The A^3 AOI consists of 1) a large FOV corresponding to the player's FOV that includes both far and near area, and 2) a small circular AOI within a close area around the player. If the player turns abruptly, the only entities that would be affected are the distant ones. The authors argue that this is a compromise way to solve the problem and is acceptable for most games.

According to classifications in [25], the A^3 uses flat architecture. Its P2P overlay is unstructured. The AOI is continuous and its shape is a combination of circle and FOV.

AOI-Cast

AOI cast is a fundamental operation in DVE to disseminate messages to other players within AOI. Jiang *et al.* [74] propose two AOI-cast schemes, called VoroCast and FiboCast, to improve the AOI scalability of P2P-based DVEs. Another AOI cast strategy for P2P DVE to minimize the traffic on the overlay is proposed by Ricci *et al.* [110]. All the above schemes are based on unstructured P2P architecture. The focus is to build an overlay according to spatial proximity of the peers. In both cases, peers are connected to a subset of their neighboring peers. The connected peers may update events such as position changes and entrance of new peers to each other.

According to classifications in [25], the above schemes use hierarchy architecture. Their P2P overlays are unstructured. The AOI is discrete and circular.

Gossip

Carlini *et al.* [26] propose epidemic (or gossip) protocols [73] to support interest management based on a hybrid architecture. Peers communicate through the gossip protocol to obtain entities' positions when they get close to each other. To reduce the load on the server, each peer dynamically maintains connections with a subset of the peers in its AOI. Thus, this method is suitable for crowded region where the AOIs of the peers are likely to be overlapped.

According to classifications in [25], the scheme uses flat architecture. Their P2P overlays are unstructured. The AOI is continuous and its shape is unspecific.

2.2.3 IM Evaluation

So far, many IM schemes have been proposed and implemented. To compare them, evaluation criteria are required. In the following paragraphs, we focus on evaluations of both C/S and P2P IM schemes.

Boulanger *et al.* [19] compare the performance of eight IM algorithms in C/S MMOG using real player data. Their selected IM algorithms include three spatial-based IM and five visibility-based IM. The first three schemes simply consider a circular AOI around the player. The AOI regions in the three schemes are represented by Euclidean distance, square tiles and hexagons respectively. The other five further consider the visibility of player for their AOI regions (e.g., ray visibility and triangulation of world space). The schemes are implemented within *Mammoth* framework [89], which is a 2D VE developed by McGill University. The experiments mainly focus on the following aspects: 1) the tile (square) size versus CPU load; 2) the effectiveness of relevance filtering versus the scalability; and 3) real-player traces versus randomly generated traces.

Zou *et al.* [142] present a detailed comparison between a seamless zone-based IM scheme and an aura-based IM scheme. Extensive simulation study is done to evaluate the cost of leaving/joining multicast groups as well as the filtering precision. The study examines the variations of the following parameters on the overall cost of the system: 1) static versus dynamic game simulation system; 2) cell sizes; 3) activeness of a scenario (a highly active scenario requires more frequent state updates). The cost is estimated through total number of joins and total number of message recipients.

Rak and Van Hook [36] evaluate region-based IM under multicast using the real data from military. The experiments use different parameters including zone size, zone alignment, the number of multicast groups and AOI shapes. They find a certain range of zone size can provide optimal results in terms of filtering precision and multicast group join rates. However, the discovery is limited to specific scenarios and simulation settings.

Fiedler *et al.* [43] compare hexagonal tiling with rectangular tiling using randomly generated movement of players. Their results show that the average number of subscribed interaction channels is smaller when using hexagons. They also demonstrate that using smaller tiles leads to the subscription of a smaller percentage of the world and receiving of a smaller number of events.

Funkhouser [47] proposes a server-based visibility algorithm called RING to compute potential visual interaction between entities, aiming at reducing the number of messages required to maintain state consistency. He compares this visibility-based approach with full message broadcast approach. The key metric used in the evaluation is the message processing rate measured by the total number of messages input/output on a single server per unit time.

Above evaluations are from recent research publications. Most of them focus on the number of messages transmitted. An evaluation aiming at cheating issues is unfortunately missing.

2.3 Cheat and Mitigation

Cheat is a common issue in online games. The game operator who fails to defeat cheaters usually suffers from decreased game subscribers. In this section, we provide a study towards the cheating issues of MMOGs, including but not limited to IE attacks.

2.3.1 Existing Cheats

In the following, we classify the existing cheats with regard to their purposes.

Access of Unentitled Information

Information Exposure (IE) attack refers to the player's behavior of gaining access to information that he or she is not entitled to. The cheat is common as developers often wrongly assume client software is trusted. In fact, a modified game client or an external cheating program is able to extract sensitive information from memory. One of the most famous examples of IE attack is the map hack in Real Time Strategy (RTS) games, such as Warcraft III [132]. To prevent the attack, sensitive information can be stored on a trusted party and loaded to the client when necessary. However, storing sensitive information on a trusted party voids P2P architecture's benefits such as scalability and reduced network latency.

Collusion

Collusion is defined as the situation where two or more cheaters collaboratively gain unfair advantages. The involved cheaters are called colluders. They can communicate with each other via an external channel, such as over the phone or instant messaging, etc. This type of cheating is very difficult to detect and prevent. A common example of collusion is that players participate in an all-against-all style match, where two cheaters team up against the others.

Operation Enhancement

Bots/reflex enhancers target at specific game such as WoW Glider [69]. The enhancer uses artificial intelligence to control the player's avatar for certain tasks, e.g., automatically shooting at other opponents. The enhancers usually deploy a proxy between the server and the client. Commands are modified when passing through the proxy to boost cheating player's actions. This type of cheat is implemented at the infrastructure level. Quake [119] is one of the first games suffering from aiming proxies [106].

Power Enhancement

Power enhancement allows a cheater to gain power beyond the game rules via sending invalid commands. The cheat is usually achieved by modifying the client program, and results in sending commands that are impossible with an unmodified game client. Such commands could give cheater's avatar enhanced power, such as extra strength or speed. Examples of this cheat happen in console games Gears of War [122].

History Changing

Suppressed update exists due to dead-reckoning strategy [11] which is applied in most networked games. Assuming dead-reckoning allows up to n consecutive packets to be dropped, a cheater can suppress first $n - 1$ consecutive updates on purpose, while still receives the updates from his opponents. Since the cheater has more information than his opponents, he/she can choose an optimal reaction. This cheat can happen in either C/S or P2P games which use dead reckoning.

Fixed delay is a type of cheat that adds a fixed amount of delay to all outgoing packets. This results in the local player receiving updates faster than that for opponents. Such cheat can lead to dramatic impact on the outcome for fast-paced games. Fixed delay usually happens in P2P games.

Timestamp attack happens in the games that require timestamps on messages. A cheater can deliberately modify the timestamp on an outgoing message after receiving updates from his/her honest opponents. Different timestamps make the server simulate

these events in different orders. Therefore, the cheater can choose the value that could mostly gain advantages in the game.

Undo is a type of cheat with regard to the use of commit/reveal scheme in some P2P protocols [11, 48, 55]. If the reveal step is not enforced, a cheater can reveal their opponents' actions and decide whether his/her committed action is optimal. If not, the cheater simply undoes the action by not revealing it.

Misleading

Inconsistency is a type of cheat that a cheater intentionally causes state inconsistency among different players by sending different game states to them. The cheater may thus gain an unfair advantage over honest players. In some cases, the cheat leads to an honest player having his game state corrupted, and thus being removed from the game.

Replay is a type of cheat that a cheater receives digitally encrypted/signed copies of updates of an opponent and re-sends them at a later time to other opponents to disadvantage them [55]. The receiver cannot recognize the true/false of the received updates because they are correctly encrypted or signed. Many games are vulnerable to such kind of cheats.

Blind opponent is a type of cheat that a cheater stops sending updates to opponents on purpose. This attack blinds opponents about the cheater's actions, while the cheater still receives updates from them [134]. This cheat is only possible in some P2P protocols [103].

Money

Real money transaction (RMT) is a transaction behavior of a game player when he purchases a game item or virtual currency using real-world currency [85]. Some MMOGs allow RMT but others consider it as cheating. A phenomenon related to RMT is gold farming, where people work full time playing MMOG to gain valuable items and sell them to players. RMT happens outside the game, often on auction sites like eBay or on dedicated message boards [125]. The cheat happens in most MMOG, among which WoW

is one of the biggest victims suffered from gold farmers, due to its quantitative numbers of players [24].

Power levelling [85] is a cheat similar to RMT, where a cheater pays another person for playing his/her character. In a role-playing-game (RPG), this type of cheats is very common. Its objective is to gain experiences and virtual items.

2.3.2 Cheating Classification

Pritchard [106] proposes a review of cheating and its prevention methods. In his article, he shares his experience as one of the developers of *Age of Empires* as well as a game player. He discusses some specific real-world cheating methods and presents practical methods to prevent them. He points out that many solutions for cheating issues do not actually prevent cheating, but make it much more difficult for players to cheat. He argues that raising the cheating cost is practically useful to stop players from cheating in the game.

Yan *et al.* [136] propose a classification of cheating in online games, specifically focusing on online game Bridge (a four-person card game). The authors also review existing methods to prevent collusion between game players. The result shows that existing methods hardly prevent collusion in online games completely. However, since Bridge is based on C/S architecture, the classification in this paper may not be suitable for P2P architecture.

Kabus *et al.* [76] discuss three types of techniques that can be used in P2P architecture to detect cheating or prevent cheating from happening. The techniques are mutual checking, log auditing and trusted computing. The first approach, mutual checking, validates player's actions before the game state modification is committed through voting from multiple randomly selected clients. The second approach, log auditing, allows the game to detect cheating when it happened, and provides a rollback mechanism for the

game to undo the effect of the cheating. The third approach, trusted computing, applies hardware-level security to prevent certain types of cheating, such as modifying the game or running cheating programs.

Neumann *et al.* [99] argue that confidentiality, integrity and availability are necessary properties for a fair game. The cheating is classified based on which property is violated. In their work, confidentiality refers to that state information is only entitled to who has the right to access. Violation of confidentiality means obtaining access to unentitled information. Integrity refers to the integrity of the game state should be maintained. Violation of integrity indicates the cheater can modify the game state without following the game rules. Availability refers to that the entire game is available to all players at all times. Violation of availability means sabotaging state updating or other functions. This paper focuses more on describing the required properties rather than complete cheating classification.

GauthierDickey *et al.* [48] classify cheating from four levels: game, application, protocol and network. Game cheats happen within the game entirely and do not modify the game or use external programs. Application cheats modify the game or use other programs. Protocol cheats interfere with the game's communication protocol. Network cheats modify the network infrastructure which affects network traffic generated by the game. The authors summarize nine types of cheats which are: denial of service, fixed delay, timestamp, suppressed update, inconsistency, collusion, information exposure, bots/reflex enhancers and breaking game rules.

Webb *et al.* [133] improve the classification in GauthierDickey's work [48]. The authors present a complete review for known cheats with the consideration of the games on both C/S and P2P architectures. In specific, it re-classifies cheating into game, application, protocol and infrastructure levels with slightly different definitions. Game-level

cheats happen within the game program but do not modify or affect the program itself. Examples of game-level cheats are bugs, real money transactions and power levelling. Application-level cheats either modify the game program or data files, or read from/write to the running game's memory. Information exposure, bots/reflex enhancers and invalid commands are typical application-level cheats. Protocol-level cheats target at network packets sent and received by the game. In the paper, nine protocol-level cheats are described: suppressed update, fixed delay, inconsistency, timestamp, collusion, spoofing, replay, blind opponent, and undo. Infrastructure-level cheats modify the software (such as display drivers) or hardware (such as the network infrastructure) that the game is using. Information exposure and proxy/reflex enhancers are two examples of infrastructure-level cheats. Note that, some cheats fall into multiple levels, such as information exposure and reflex enhancers. This paper is also where the Information Exposure (IE) attack is originally defined.

2.3.3 Mitigation on Cheats

Many protocols have been proposed to prevent or mitigate cheating issues. Note that some protocols, such as PunkBuster [40] and Valve Anti-Cheat [128], can be used in both C/S and P2P architectures. Others are limited to specific architecture.

2.3.3.1 C/S Cheat Mitigation

On Demand Loading

In the C/S architecture, On Demand Loading (ODL) [86] is the most effective solution to prevent the private information from unauthorized disclosure. A trusted third party (often the server) stores all confidential information and only transmits the information to entitled clients. By doing this, the exposure of information is not going to happen. However, this method is hard to be ported to the P2P architecture, especially for IM.

PunkBuster

Another approach to defeat cheats is to deploy cheat detection applications such as PunkBuster [40]. Like anti-virus applications, the cheat detection application can verify the integrity of installed game program and detect known cheating programs or other threats. However, this approach is based on the assumption that the cheat detection application is not sabotaged and the cheating program or threat is previously known, which may not be the case all the time.

Valve Anti-Cheat

The Valve Anti-Cheat (VAC) system is another cheat detection system which detects cheat. VAC-secured server will ban a player from playing the game on the server if the player installs an identifiable cheating program on his computer. Identifiable cheats refer to any third-party modifications to the game aiming to give the player an advantage over others. For example, changing of a game's core executable files and dynamic link libraries is classified as a cheat.

2.3.3.2 P2P Cheat Mitigation

Other work has been proposed to solve cheating issues in P2P game. However, existing research mostly focuses on active attacks (e.g., inconsistency) that are detectable and recoverable [33, 48, 55, 134, 135].

Age of Empires

Age of Empires (AoE) is one of the first commercial P2P games [106]. The protocol of AoE prevents invalid command and information exposure attacks. To prevent invalid commands, peers perform mutual checking. The protocol agrees that all invalid commands are dropped by the receiver. In addition, peers exchange the hashes of their game state periodically. By doing so, the cheater can be detected because his game state is different from other players' states. To prevent information exposure, AoE encrypts all the data from the game in memory, and decrypts them only when performing operations.

It is a mitigation mechanism that makes the cheater locate game data more difficult, though it cannot entirely prevent the information exposure attack.

Referee Anti-Cheat Scheme

Referee Anti-Cheat Scheme (RACS), proposed by Webb *et al.* [134], is a P2P-based solution to enhance the scalability of C/S model without decreasing security. In particular, it provides a mechanism to defeat IE attacks. RACS assumes a hybrid architecture of C/S and P2P. In P2P mode (denoted as PP mode in RACS), it allows peers to communicate directly and thus improves its scalability. To prevent cheats, the referee (i.e., the server in the traditional C/S architecture) acts as an arbitrator to monitor peers' behavior. If the referee suspects a peer is cheating, it will revert that peer from PP mode to Peer-Referee-Peer (PRP) mode which only allows the peer exchange updates with the referee. To eliminate information exposure, private information has to be transmitted through referee. However, in the PRP mode, the referee has to receive and process peers' updates like the server does in the C/S architecture. This reduces the scalability provided by the P2P architecture. In addition, forcing all private information to be transmitted through the referee will inevitably increase the communication latency.

Lockstep and Sliding Pipeline

Lockstep [10] is a protocol that divides game time into rounds and requires in the game that every player submits his/her move within the current round before the next round begins. The commit/reveal mechanism is applied to prevent cheating. Lockstep can be proven secure against all protocol-level cheats except inconsistency cheat [133], because it does not use digital signatures to authenticate updates. Lockstep is also too slow to be accepted for many fast-paced games. In the worst case, delay can be up to $3d$, where d is the delay between two slowest players.

The Sliding Pipeline (SP) protocol [35] is developed based on lockstep. It allows techniques of pipelining and dead reckoning to be used for updates, and thus improving the smoothness of the game. SP also constantly monitors the delay between players, in order to determine the maximum delay allowable for an update. It thus solves the fixed delay, timestamp, suppressed update cheats and blind opponent cheats. However, SP cannot distinguish between players suffering from slow network and cheaters.

NEO

The New Event Ordering (NEO) protocol [48] is an effective method in limiting malicious behavior to a predefined level. NEO explicitly defines this limit to be $2d$ per round (d is the delay between two slowest players as defined in SP protocol). Players must send their updates within the first half of the round, and transmit their keys in the second half of the round. Only within the referred time are the updates accepted as valid. To increase responsiveness, the rounds can be pipelined. Although the predefined limitation is $2d$, NEO also provides mechanisms to redefine the round and improve the responsiveness.

SEA/SGA

The Secure Event Agreement (SEA) protocol [55] is an improved version of NEO by modifying cryptographic techniques. The authors of SEA argue that NEO suffers

from several cheats such as replay attack, spoofing, and inconsistency. They propose a protocol called SEA to prevent these attacks. However, SEA is vulnerable to undo cheat.

The Secure Group Agreement (SGA) protocol [33] is a complement of SEA. SGA divides peers into groups. Within a group, peers use SEA to agree on events in the game. When certain percentage of corrupt members is determined, the SGA protocol provides a selection mechanism to form groups.

2.4 Summary

In this chapter, we have reviewed the details of network architectures, interest management and cheating issues in MMOGs. Specifically, we have compared C/S and P2P architectures, and listed the challenges that P2P architecture is facing. For IM, we have reviewed classification, definition and evaluation towards the existing IM schemes. We have also given a study towards the cheating issues of MMOGs, including but not limited to IE attacks. As far as we know, there is no existing work on mitigating IE attacks on the IM component, especially for P2P games. We must stress that different interest management mechanisms synchronize the publishers game states to the subscriber at different conditions. However, as long as sensitive information in plaintext is transferred during the interest management, information exposure attacks will exist. In the next chapter, we will formally define the problem of mitigating IE attacks on P2P based IM schemes.

Chapter 3

Problem Statement

In this chapter, we define the problems that this thesis is going to tackle. We start by formally defining what IM is, and then focus specifically on IE attacks in IM. We also specify several important properties for the evaluation of IM schemes. Lastly, we model two existing IM schemes, CS and SP2P, using our definition.

3.1 Definition of IM

In this section, we formally define what IM is. We describe the functionality of IM that is universal to all IM schemes. For a specific IM scheme, we define it as an interactive protocol. Notations and assumptions of the protocol are given. In addition, we use a lite MMOG to explain how the IM protocol is integrated to the game.

3.1.1 Functionality of IM

IM is a vital component in MMOG. It involves a state publisher (publisher or pub) and a state subscriber (subscriber or sub). The functionality of IM is to reveal the publisher's game state (gs_{pub}) to the subscriber if and only if the game state is relevant to the subscriber. This relevance is defined as whether the publisher's position (pos_{pub}) is contained within the subscriber's AOI (aoi_{sub}). We can model the functionality of IM

as function f_{im} :

$$f_{im}(gs_{pub}, aoi_{sub}) = \begin{cases} gs_{pub} & \text{if GETPOS}(gs_{pub}) \text{ is contained within } aoi_{sub} \\ Ud & \text{otherwise} \end{cases}$$

The inputs gs_{pub} and aoi_{sub} are the publisher's game state and the subscriber's AOI respectively. The output can be either gs_{pub} or Ud (undefined), which will be made available to the subscriber.

3.1.2 Definition of IM Schemes

The functionality of IM is universal to all IM schemes. However, different IM schemes have different ways to obtain the result of f_{im} . We define an IM scheme s as an interactive protocol $\mathcal{P}_s$ such that:

$$\mathcal{P}_s ::= \langle E, I \rangle$$

The protocol is defined as a tuple $\langle E, I \rangle$ where element E denotes the entities involved in the protocol and element I denotes the initial input sets of the entities. Possible entities that can appear in E include but not limited to the publisher pub , subscriber sub , central server srv and Trusted Third Party (TTP) ttp . For example, $E = \{pub, sub\}$ indicates that the protocol is carried out between the publisher pub and subscriber sub . Especially, the subscriber as the output receiver must participate in the protocol (i.e., $sub \in E$ is always satisfied). The input set of an entity includes IM-related states that are essential to execute the protocol. For example, $I = \{in_{pub} = \{gs_{pub}\}, in_{sub} = \{aoi_{sub}\}\}$ indicates that the publisher initially knows its game state gs_{pub} and the subscriber initially knows its AOI. The protocol defines how the entity $e \in E$ should behave in the protocol (a.k.a., benign behavior). The behavior includes calculation based on the known information (including input set in_e), and information exchange with other entities. It is usually described by pseudo code or sequence charts. We say $\mathcal{P}_s$ is faithfully performed if and only if every entity $e \in E$ follows the protocol with its genuine in_e as the input. In this case, entity sub should finally get the value identical to the result of $f_{im}(gs_{pub}, aoi_{sub})$.

Table 3.1: General notations for modeling IM schemes

pos_x	Peer x 's position
aoi_x	Peer x 's AOI
er_x	Peer x 's ER
gs_x	Peer x 's game state
V	Virtual environment
$p \in R$	Returns <i>true</i> iff point p is contained by region R
$R_1 \subset R_2$	Returns <i>true</i> iff region R_1 is fully covered by region R_2
$R_1 \cap R_2$	Returns a new region $R_3 = \{e \mid e \in R_1 \wedge e \in R_2\}$
$R_1 - R_2$	Returns a new region $R_3 = \{e \mid e \in R_1 \wedge e \notin R_2\}$
$GETAOI(pos_x)$	Returns an AOI constructed by position pos_x
$GETER(pos_x)$	Returns an ER constructed by position pos_x
$GETPOS(aoi_x)$	Returns the position used to construct the AOI
$GETPOS(er_x)$	Returns the position used to construct the ER
$GETPOS(gs_x)$	Returns the position described by the game state
$ISGEN(imst_x)$	Returns <i>true</i> if $imst_x$ is a genuine IM-related game state of peer x

3.1.3 Notations and Assumptions

In this thesis, a *peer* refers to either the publisher or the subscriber, a logic entity or a party who participates in the protocol. A *player* refers to a person who plays the game. We assume a peer controls only one avatar in the game. In most cases, words peer, player and avatar can be used interchangeably.

According to terminologies of the Distribution Data Management (DDM) in High Level Architecture (HLA) [88], a Virtual Environment (VE) corresponds to a routing space. That is, a multidimensional coordinate system by which each peer can express his/her interest for either sending or receiving data. Informally speaking, VE is the virtual world that a player can explore. In our work, we define VE as a finite region with discrete coordinates. The position of a player can be a point in this coordinate system. Similar definitions can be found in [129, 140].

Table 3.1 summarizes typical notations used in this thesis. Variables pos_x , aoi_x , er_x and gs_x denote peer x 's position, AOI, ER and game state respectively. Variable V denotes the VE. Since VE is defined as a finite region with discrete coordinates, it can be

modeled as a finite set V where each coordinate in the VE is modeled as an element in the set. Regions such as AOI and ER can also be modeled as finite sets and are subsets of V . In this case, operations defined in set theory can be reused in the context of IM. For example, operation $p \in R$ tests if point p is contained by region R . Operation $R_1 \subset R_2$ tests if region R_1 is fully covered by region R_2 . These two operations can be used as Boolean functions. Similarly, $\cap$ and $-$ are defined for operations between regions.

We assume peers/avatars are the only objects in the VE. An avatar's AOI/ER can be constructed by using its position. Hence, knowing an avatar's AOI/ER implies its position can be derived accordingly. In addition, an avatar's position can be extracted from its game state. Related functions are defined as $\text{GETAOI}(pos_x)$, $\text{GETER}(pos_x)$, $\text{GETPOS}(aoi_x)$, $\text{GETPOS}(er_x)$ and $\text{GETPOS}(gs_x)$. The shape and size of AOI/ER are assumed known and fixed during the game. Particularly, we assume peer or TTP owns a function $\text{ISGEN}(imst_x)$ that can testify whether $imst_x$ is a genuine IM-related state of player x , where $imst_x$ can be pos_x , aoi_x , er_x , or gs_x . The assumption is based on the fact that the peer who receives the IM-related state $imst_x$ can detect whether $imst_x$ is inconsistent to its replicas, other received states or game rules. This inconsistency detection is available even in P2P architecture [48, 55, 76]. The function $\text{ISGEN}(imst_x)$ will be used in IM schemes for detecting forged states.

The communication channels between any two entities are considered as reliable and secure. The channel satisfies the following properties.

- No messages are lost during the transmission.
- A message is delivered only once.
- If a peer receives a message, this message must be sent from the other peer.
- If the receiver uploads a received message to TTP, TTP can verify whether this message is sent from the specific sender.

- (i) Alice and Bob move their positions in the game.
- (ii) Alice and Bob perform $\mathcal{P}_s$ with $sub = bob$, and $pub = alice$ (if $pub \in E$).
- (iii) Bob and Alice perform $\mathcal{P}_s$ with $sub = alice$, and $pub = bob$ (if $pub \in E$).
- (iv) Repeat step (i).

Figure 3.1: The behaviors of Alice and Bob in the lite game

- The message is resistant to tampering. That is, any messages tampered during transmission will be detected and discarded.
- The message contents are kept secret from any third party.

3.1.4 A Lite Game

As a component in MMOG, IM cannot run alone. To evaluate it, we must study its behavior when it is integrated into a game. To this end, we model a lite game that only performs essential interactions required by IM. Considering Alice (*alice*) and Bob (*bob*) are two peers in the lite game, their behaviors can be modeled in Figure 3.1. In step (i), both Alice and Bob move their (avatars') positions. This leads to the change of their positions, AOIs, ERs (if any) and game states. In step (ii), Alice and Bob act as the publisher and subscriber respectively to perform the IM protocol $\mathcal{P}_s$. Bob learns the output of the protocol, which should be $f_{im}(gs_{alice}, aoi_{bob})$. In step (iii), Alice and Bob swap their roles and perform the protocol again. That is, Bob and Alice act as the publisher and subscriber respectively to perform the IM protocol $\mathcal{P}_s$. This time, Alice learns the output of the protocol, which should be the result of $f_{im}(gs_{bob}, aoi_{alice})$. Steps (i) to (iii) are executed repeatedly and infinitely. The lite game is a simplified case of MMOG. However, since the behavior of IM between a pair of peers is independent to what between to other pairs, the conclusions drawn on the lite game can be extended to more generic situations.

Figure 3.2: Protocol of $\mathcal{P}_{cs}$

3.2 Modeling CS and SP2P Schemes

In this section, we model two existing IM schemes CS and SP2P as examples. These two IM schemes are typical IM schemes on C/S and P2P architectures respectively. We demonstrate the effectiveness and usefulness of our definition of IM scheme as a protocol through these examples.

3.2.1 CS

We name the C/S based IM scheme mentioned in Chapter 1 as ‘CS’. The protocol $\mathcal{P}_{cs}$ is defined as

$$\begin{aligned}
 \mathcal{P}_{cs} &::= \langle E, I \rangle \\
 E &= \{srv, sub\} \\
 I &= \{in_{srv}, in_{sub}\} \\
 in_{srv} &= \{pos_{pub}, gs_{pub}, aoi_{sub}\} \\
 in_{sub} &= \emptyset
 \end{aligned}$$

The protocol is performed between the server srv and subscriber client sub ($E = \{srv, sub\}$). Since the protocol is performed on C/S architecture, the server srv initially knows the publisher’s position pos_{pub} and game state gs_{pub} , and the subscriber’s AOI aoi_{sub} ($in_{srv} = \{pos_{pub}, gs_{pub}, aoi_{sub}\}$). Because of this, the publisher does not need to participate in the protocol to explicitly provide its position and game state. The subscriber initially knows nothing ($in_{sub} = \emptyset$). The protocol is illustrated in Figure 3.2.

From the initial input set in_{srv} , the server can calculate whether the publisher's game state is relevant to the subscriber by computing $rel = (pos_{pub} \in aoi_{sub})$. if $rel = true$, the server sends the publisher's game state gs_{pub} as game state update message msg_{gs} to the subscriber. The subscriber, if receives msg_{gs} , can recover the publisher's game state gs'_{pub} as the protocol output. Otherwise, the subscriber outputs Ud . In this protocol, the only message transmitted over the Internet is msg_{gs} .

3.2.2 SP2P

In Chapter 1, we have illustrated a widely used P2P based IM scheme—Standard P2P (SP2P). Compared with the CS scheme, ER is additionally employed in SP2P as an approximation of the subscriber's AOI, and is compared with the publisher's position. This approximation is commonly required for IM schemes in P2P network architecture for optimizing network performance.

The SP2P protocol can be defined as follows.

$$\begin{aligned}
 \mathcal{P}_{sp2p} &::= \langle E, I \rangle \\
 E &= \{pub, sub\} \\
 I &= \{in_{pub}, in_{sub}\} \\
 in_{pub} &= \{pos_{pub}, gs_{pub}, cer_{sub}\} \\
 in_{sub} &= \{pos_{sub}, er_{sub}\}
 \end{aligned}$$

The protocol is performed among the publisher pub and subscriber sub ($E = \{pub, sub\}$). The publisher initially knows its position, game state and the subscriber's ER since last change ($in_{pub} = \{pos_{pub}, gs_{pub}, cer_{sub}\}$). The subscriber initially knows its position and ER ($in_{sub} = \{pos_{sub}, er_{sub}\}$). The protocol is defined in Figure 3.3. It can be divided into two stages, SL update stage (step (i)(a) to step (ii)(b)) and game state update stage (step (ii)(c) to step (iii)(b)). The SL update is to update the publisher's subscriber list by comparing the subscriber's ER with the publisher's position. To this end, the subscriber must update its ER to the publisher via ER update message (msg_{er}) when the ER is changed. On the publisher's side, state cer_{sub} is employed to cache subscriber's ER and

```
/* Begin of SL update */  
  
(i) The subscriber  
  (a) computes  
         $aoi_{sub} = \text{GETAOI}(pos_{sub})$   
         $aoi_{iner} = (aoi_{sub} \subset er_{sub})$   
  (b) if  $aoi_{iner} = false$ ,  
        (1) updates  $er_{sub} = \text{GETER}(pos_{sub})$ ;  
        (2) sends  $msg_{er} = er_{sub}$  to the publisher.  
  
(ii) The publisher  
  (a) if  $msg_{er}$  is received,  
        (1) stores  $msg_{er}$  as  $cer_{sub}$ ;  
        (2) aborts the protocol if  $\text{ISGEN}(cer_{sub}) = false$ .  
  (b) computes  $insl = (pos_{pub} \in cer_{sub})$   
        /* End of SL update (begin of game state update) */  
  (c) if  $insl = true$ , sends  $msg_{gs} = gs_{pub}$  to the subscriber.  
  
(iii) The subscriber  
  (a) if  $msg_{gs}$  is received,  
        (1) stores  $msg_{gs}$  as  $gs'_{pub}$ ;  
        (2) aborts the protocol if  $\text{ISGEN}(gs'_{pub}) = false$ ;  
        (3) computes  
               $pos'_{pub} = \text{GETPOS}(gs'_{pub})$   
               $rel = (pos'_{pub} \in aoi_{sub})$   
        (4) if  $rel = true$ , outputs  $gs'_{pub}$ ; otherwise, outputs  $Ud$ .  
  (b) otherwise, outputs  $Ud$ .  
  
/* End of game state update */
```

Figure 3.3: Protocol of $\mathcal{P}_{sp2p}$

keep it updated. If received ER is not genuine, the publisher will abort the protocol. This cached ER is compared with the publisher's position for the SL update. The result of SL update is Boolean value $insl$ which indicates whether or not the subscriber is on the

publisher's SL. The game state update is to let the subscriber know the publisher's latest game state. If the subscriber is in the SL ($insl = true$), the publisher will send its game state update message (msg_{gs}) to the subscriber. If msg_{gs} is received, the subscriber will save it as gs'_{pub} if received game state is genuine. The subscriber will further compute whether received game state is relevant to himself/herself. In specific, the subscriber extracts the publisher's position from the game state ($pos'_{pub} = GETPOS(gs'_{pub})$) and computes whether this position is inside its AOI ($rel = (pos'_{pub} \in aoi_{sub})$). If relevant, the protocol will output gs'_{pub} ; otherwise, it will output Ud .

3.3 Definition of IE Attacks

The terminology 'IE attack' is defined by its purpose, i.e., information exposure. To study this attack, we must model its behavior which results in the information exposure.

3.3.1 Adversary Model

The behavior of an entity in executing a protocol can be classified as *honest*, *Honest-But-Curious (HBC)* and *malicious*. An honest entity always faithfully executes the IM protocol. Except the protocol output to himself/herself, the honest entity will not obtain any additional information from received messages or intermediate results in the protocol. Especially, we assume the central server srv and TTP ttp are always honest. In the P2P protocols proposed by this thesis, TTP is introduced as a verifier or arbitrator. In cryptography, TTP is an entity trusted by all parties to facilitate verification and arbitration. As TTP behaves honestly, it helps peers to verify states and arbitrate critical conflicts. The difference between the central server and TTP is that TTP should only handle light-weight and less frequent tasks such as arbitration.

On the contrary, peers in a P2P architecture can also be HBC or malicious. An HBC peer also faithfully follows each step of the IM protocol. However, an HBC peer may

obtain the information that passes to him/her, such as received messages or intermediate results, and try to learn as much as possible from the information that he/she has known. From the view of other peers in the protocol, the HBC peer acts exactly the same as an honest one does. However, the HBC peer may know more information than the honest peer does.

A malicious peer may not follow the protocol. The malicious peer may forge the input to the protocol (e.g., in_{pub}), modify any steps of the protocol and/or send compromised messages to other peers. In other words, the malicious peer may send a message whose content is different from what it should be when the protocol is performed by an honest peer. Because of this, malicious behavior might be detected by the other party in the protocol. In the context of IM, we assume a message with forged IM-related state can be detected. This detection is provided by $ISGEN(imst_x)$ as mentioned in the last section. This function will return *true* if $imst_x$ is a genuine state of peer x ; otherwise if $imst_x$ is a forged state, the function will return *false*. With this function, it might be easy to detect malicious behavior when $imst_x$ is received in plaintext. But in some IM schemes, one may receive an obfuscated message which is generated from a forged state. In this case, simply applying function $ISGEN$ to the received message does not work and the IM scheme must seek other approaches to detect the forged state.

Note that, in most existing definitions of malicious behavior [51], forging input is excluded from the attacker's capabilities. However, in the context of network games, forging input commonly happens and has considerable impact to the victim, e.g., more information exposure. Therefore, we assume a malicious attacker is capable of forging input and an IM scheme should detect this attack.

We stress that our goal is to mitigate IE attacks on IM. In specific, mitigating HBC attacks requires the IM scheme to prevent the attacker from knowing the sensitive information. Mitigating malicious attacks requires the IM scheme to allow the victim to detect the attack. A good IM scheme should mitigate both HBC and malicious attacks.

3.3.2 IE Attacks

Using the adversary model, the IE attack is defined as the behavior of HBC or malicious peer that especially targets on exposing sensitive information. The sensitive information is specifically defined as one's position. The HBC or malicious peer is called attacker, and the other benign peer is called victim. When the attack is based on malicious behavior, it is called 'effective' if and only if the attack cannot be detected in any cases, i.e., the detection probability is zero. In our work, we assume there is only one attacker in the game.

3.4 Properties of IM Schemes

In this section, we specify several properties for IM protocols in order to evaluate them. The properties include correctness, secrecy and bandwidth usage. Correctness evaluates whether $\mathcal{P}_s$ is functional. The mitigation to IE attacks is assessed by secrecy. Especially, secrecy-1 verifies whether the adversary knows the victim's exact position or not. Secrecy-2 quantitatively evaluates how much knowledge the adversary has on the victim's position. As the original purpose of IM is to reduce unnecessary communications, the evaluation of bandwidth usage is also included. These selected properties cover the core functionality and network performance of IM, as well as its ability to mitigate IE attacks. We believe these properties can effectively and sufficiently evaluate IM schemes.

3.4.1 Correctness

An IM scheme must fulfill the functionality of IM correctly. That is, the protocol $\mathcal{P}_s$ is supposed to output the result of $f_{im}(gs_{pub}, aoi_{sub})$ in every round when both Alice and Bob are honest or HBC. The correctness property is defined as follows.

Correctness The output of $\mathcal{P}_s$ is identical to the result of $f_{im}(gs_{pub}, aoi_{sub})$.

In this thesis, correctness property is formally verified via model checking. The evaluation process and results will be detailed in Chapter 6.

3.4.2 Secrecy-1

Suppose we regard knowing the exact position of the victim as a successful IE attack, secrecy-1 defines the situation when the attack is defeated. With Alice as the attacker and Bob as the victim, the secrecy-1 property is defined as follows:

Secrecy-1 Alice knows the Bob's position only if Alice acts as the subscriber and

$$f_{im}(gs_{bob}, aoi_{alice}) = gs_{bob}.$$

In other words, the secrecy-1 will be violated if 1) the publisher knows the subscriber's position, or 2) the subscriber knows the publisher's position but the publisher's game state is irrelevant to the subscriber. The result of property evaluation is binary, either satisfied or unsatisfied. In this thesis, secrecy-1 of $\mathcal{P}_s$ is also formally verified via model checking and will be detailed in Chapter 6.

3.4.3 Secrecy-2

Although secrecy-1 is straightforward, the property is limited to its binary result, which fails to evaluate the mitigation to IE attacks in a finer level. For example, if the attacker knows the victim is inside a smaller region, even though the attacker does not know the exact position of the victim, the attacker has more information to predict where the victim is than only the situation when he/she only knows the victim is in the VE. The limitation of secrecy-1 leads us to define secrecy-2.

Secrecy-2 Alice predicts Bob's position as a random variable X and secrecy-2 is defined as the average Shannon entropy of X per round.

In the game, Alice may not know the exact position of Bob. However, she can predict in what probability Bob is at a certain position. The position of Bob is modeled as a random variable X by Alice with defined Probability Mass Function (PMF)

$$f_X(p) = Pr(X = p)$$

which predicts the probability of ‘Bob is at position p ’ ($Pr(X = p)$) for all possible p over the entire VE. The Shannon entropy of X in round i is thus calculated as

$$entropy_i = - \sum_{p \in V} f_X(p) \log_2 f_X(p) \quad (\text{bits})$$

and the secrecy-2, or average entropy per round can be calculated as $\overline{entropy}$.

The Shannon entropy is a standard way to measure uncertainty. In our case, the entropy represents the randomness of X , or how unsure Alice is about where Bob is. The entropy is a number above or equal to zero. The zero entropy is reached if and only if Alice knows the exact position of the victim. The higher the entropy, the better mitigation to IE attacks. In this thesis, secrecy-2 of $\mathcal{P}_s$ is evaluated via simulation, which will be detailed in Chapter 7.

3.4.4 Bandwidth Usage

Since the original purpose of IM is to reduce unnecessary communications, the network performance of an IM scheme is vital. Therefore, we evaluate the average bandwidth usage per round of $\mathcal{P}_s$. For an IM scheme, bandwidth usage is calculated based on the number of messages transmitted among entities and the length of each message. Note that, we only evaluate the communications over the Internet. In this thesis, bandwidth usage evaluation is by simulation, which will be detailed in Chapter 7.

3.5 Summary

In this chapter, we have formally defined the problems this thesis focuses on. We have defined an IM scheme as an interactive protocol. We have also defined the adversary behavior in the context of IM. Properties such as correctness, secrecy-1, secrecy-2 and bandwidth usage are presented as evaluation criteria of IM schemes. As examples, two existing typical IM schemes are defined as interactive protocols. We have demonstrated the effectiveness of defining IM schemes as protocols through these examples. Beginning from the next chapter, this thesis will focus on the discovery of new IM schemes, aiming to mitigate IE attacks.

Chapter 4

SAFE

In this chapter, we propose a new IM scheme called Secure Area of interests FiltEr (SAFE) to mitigate information exposure for interest management. The scheme, defined as protocol $\mathcal{P}_{safe}$, includes a main protocol which adopts privacy preserving computation to mitigate IE attacks, and supplementary commitment protocols to defeat malicious attacks. Specifically, $\mathcal{P}_{safe}$ aims to guarantee that only the result of the SL update stage (i.e., *insl*) can be revealed to the publisher. Since privacy preserving protocol can be implemented on software, SAFE becomes a widely adaptable IM scheme for most existing platforms.

We organize the following sections in this way: Section 4.1 discusses our design rationale behind the SAFE protocol. Section 4.2 describes the privacy preserving protocol which is the building block for securing the SL update stage. Section 4.2.2 describes the proposed commitment scheme protocols. Section 4.3 presents the complete SAFE protocol. Most work in this chapter is derived from our previously published paper ‘SAFE: A Secure Area of Interests Filter for P2P-based MMOGs’ [61].

4.1 Design Rationale

Recall that in the SL update stage of SP2P protocol $\mathcal{P}_{sp2p}$, the subscriber is required to synchronize his/her ER to the publisher, so that the publisher can perform the compar-

ison between the subscriber's ER and the publisher's position. This requirement would allow HBC or malicious publisher to learn the subscriber's position from received ER. Therefore, we are motivated to investigate methods to reveal only the result *insl* to the publisher without disclosing any party's sensitive information to each other. The functionality of SL update stage is modeled as

$$insl = f_{sl}(pos_{pub}, er_{sub}) = \begin{cases} true & \text{if } pos_{pub} \in er_{sub} \\ false & \text{otherwise} \end{cases}$$

where pos_{pub} and er_{sub} are the publisher's position and the subscriber's ER respectively. We need to find a protocol to calculate *insl* without revealing the publisher's pos_{pub} and subscriber's er_{sub} to each other.

4.1.1 Modeling f_{sl} as Set Membership Testing

One solution to securely obtain the result of a function is via a privacy preserving protocol [80]. For a function that is collaboratively calculated by two entities, each entity provides a private input and the function result is based on these inputs. The privacy preserving protocol provides a mechanism to obtain the result of the function without disclosing the entities' private inputs to each other.

If the definition of f_{sl} is kept unchanged, we can model f_{sl} as a set membership function $pos_{pub} \in er_{sub}$ with pos_{pub} and er_{sub} as two private inputs provided by the publisher and the subscriber respectively. A privacy preserving protocol for calculating the set membership can be employed to guarantee 1) the publisher will know *insl*; 2) the publisher cannot know er_{sub} ; and 3) the subscriber cannot know pos_{pub} .

Although this solution works effectively to mitigate IE attacks, it could lead to unacceptable network performance. The original design of SP2P allows the publisher to cache the subscriber's ER and therefore the publisher can calculate *insl* by itself when ER is unchanged. But the privacy preserving protocol does not reveal the subscriber's ER in plaintext to the publisher. When either the subscriber's ER or the publisher's position

is changed, the privacy preserving protocol must be re-executed to reveal *insl*. As the publisher's position is expected to change frequently, the privacy preserving protocol will be executed in a high frequency as well, which would consume excessive bandwidth.

4.1.2 Detection Region

To solve the network problem in the last subsection, a possible solution is to avoid using the privacy preserving protocol for the function that has frequently changed states (e.g., the publisher's position) as the private inputs. Therefore, we seek another way to redefine the functionality of SL update (f_{sl}) in SAFE.

Inspired by ER which is the approximation of the subscriber's AOI, we define Detection Region (DR) as the approximation of the publisher's position (i.e., $pos_{pub} \in dr_{pub}$). Similarly, the publisher's DR is defined as a region in VE, covering the publisher's position. DR is denoted as dr_{pub} and can be generated by invoking $dr_{pub} = getDr(pos_{pub})$. Like what the subscriber does for keeping its ER, the publisher should keep its DR unchanged until necessary, i.e., the publisher's position is outside DR. Therefore, instead of comparing the subscriber's ER with the publisher's position, now the SL update is based on comparing the subscriber's ER and the publisher's DR. As DR update is much less frequent than position update, the network overhead of executing the SL update is largely reduced.

In other words, we redefine the functionality of the SL update as

$$f_{sl}(dr_{pub}, er_{sub}) = (dr_{pub} \cap er_{sub} \neq \emptyset)$$

which models the SL update as a disjointness test. Note that, although f_{sl} is redefined, the functionality of game state update stage is unmodified. Therefore, even if the refined SL update may result in more irrelevant game states, the game state update will filter them out, keeping IM functionality unchanged. Now we need to find a suitable privacy preserving protocol for calculating the disjointness test.

Table 4.1: Summary of candidate privacy preserving protocols on private set operations

Protocol	Functionality	Latency (RTTs)	Bandwidth (bits)
FNP [44]	Cardinality	1	$O(S_{pub})$
HW [66]	Cardinality	1	$O(S_{pub} + S_{sub})$
KM-1 [79]	Disjointness	1	$O(\Omega)$
KM-2 [79]	Disjointness	$ S_b $	$O(S_{pub} S_{sub})$
KM-3 [79]	Disjointness	1	$\binom{ S_{pub} + S_{sub} }{ S_{sub} }$

Note that there is a tradeoff between more DR updates and more irrelevant game state updates, depending on the size of DR. Smaller DR demands more execution of SAFE and thereby consumes more network bandwidth. Larger DR leads to more IE attacks. The choice should depend on game content (e.g., whether players are moving fast) and network condition (e.g., whether high bandwidth).

4.2 Privacy Preserving Protocol

In academia, a few privacy preserving protocols have been proposed to solve two-party disjointness test problem. The disjointness test can be generalized as function

$$intersect(S_{pub}, S_{sub}) = (S_{pub} \cap S_{sub} \neq \emptyset)$$

where $S_{pub}, S_{sub} \subset \Omega$. In our case, $S_{pub} = dr_{pub}$, $S_{sub} = er_{sub}$ and $\Omega = V$ where V stands for the whole set of two private inputs, i.e., $S_{pub}, S_{sub} \subset \Omega$. Table 4.1 summarizes the functionality and network performance of commonly used protocols such as FNP protocol proposed by Freedman, Nissim and Pinkas [44] and HW protocol proposed by Hohenberger and Weis [66]. Symbols S_{pub} and S_{sub} denote the private sets for the publisher and the subscriber respectively.

However, besides disjointness, FNP and HW protocols also reveal the cardinality of the intersect of two private sets, which may expose sensitive information such as the distance between the two peers. Thus, these protocols may not fit the secrecy requirements

Figure 4.1: Algorithm of KM-1

of the SL update. Our special requirement for the privacy preserving protocol demands that only the Boolean result that whether or not two sets have common elements can be revealed. The cardinality of the intersected set has to be kept secret. Kiayias and Mitrofanova [79] proposed KM-1, KM-2 and KM-3 which satisfy our needs. Particularly, KM-1 is designed for the situation where two private sets are represented by bitstrings with limited length, which fits our scenario.

4.2.1 KM-1 Protocol

We reuse the algorithm of KM-1 as the building block for our SAFE protocol. The KM-1 protocol is based on homomorphic encryption (see Section A.1.4 for background), which is shown in Figure 4.1. Operators E and D are homomorphic encryption and decryption

functions respectively. Operator $\odot$ stands for multiplication on the encrypted numbers. The public key pk and secret key sk are securely generated homomorphic key pair. The public key pk is known to both the publisher and the subscriber, and the private key sk is only known to the publisher. The two private sets S_{pub} and S_{sub} are represented as bitstrings. For each private set S_x ($x \in \{pub, sub\}$), we define

$$bs_x = \langle bit_{x_1}, \dots, bit_{x_{|\Omega|}} \rangle$$

where $bit_{x_i} = 1$ iff $i \in S_x$.

Especially, two messages (msg_1 and msg_2) are transferred between the publisher and the subscriber, which can be modeled as message generation functions

$$msg_1 = f_{msg_1}(S_{pub}, R_{pub})$$

$$msg_2 = f_{msg_2}(msg_1, S_{sub}, n_1, n_2)$$

where $R_{pub} = \langle r_1, r_2, \dots, r_{|\Omega|} \rangle$ as a random number set. The message msg_1 (msg_2) is determined by the inputs to f_{msg_1} (f_{msg_2}). For example, msg_1 sent from the publisher to the subscriber is determined by the publisher's private set S_{pub} and locally generated random number set (R_{pub}).

It has been proved that KM-1 reveals only the disjointness to the publisher even if one of the parties involved in the protocol has HBC behavior. Moreover, the protocol requires only a single round of communication, which is a desirable feature of interactive protocols. The bandwidth usage of KM-1 is $O(|\Omega|)$.

4.2.2 Detection of Malicious Behavior

For defeating malicious attacks, the original work of KM-1 employs non-interactive zero-knowledge proof [54]. It guarantees the private inputs are strictly kept by their owners but is costly in the sense of network performance. Moreover, the zero-knowledge proof

does not detect forged input. To address this problem, we design supplementary protocols relying on commitment scheme, with low network overhead, to detect malicious behavior including forged input.

In cryptography, a commitment scheme is a protocol that allows one party to commit on a chosen value while keeping it hidden to another party, with the capability to reveal the committed value later [53]. It is designed for the purpose of avoiding the values or statements to be changed after they have been committed by the party. Often there are two parties in a commitment scheme: the party who chooses the value is the sender, and the other party who receives the value is the receiver. A metaphor of a commitment scheme is that a sender puts a message in a locked box and gives the box to a receiver. The sender cannot change the message inside the box after giving the box out, but can choose to give the receiver the key some time later. The receiver cannot read the message in the locked box unless he/she obtains the key from the sender. In particular, the ability to prevent the sender from changing the message in the box is called *hiding* property. The ability to prevent the receiver from knowing the message in the box before having the key is called *binding* property. A commitment scheme must achieve both hiding and binding properties.

A commitment scheme has *commit* and *reveal* phases. In the commit phase, a commitment message containing the chosen value is sent to the receiver, like the process of preparing the locked box and sending it to the receiver in the above metaphor. In the reveal phase, the chosen value is revealed by the sender and the receiver can verify it, like the process of giving the key to the receiver in the metaphor. In general, the receiver can be a peer or TTP.

4.2.3 Supplementary Protocol

We design two supplementary protocols for detecting malicious behavior in f_{msg_1} and f_{msg_2} respectively. Simply speaking, the supplementary protocols work as the verification

 Figure 4.2: Protocol of $\mathcal{SP}_{sub}$

process of the main KM-1 protocol. If a message generation function in the main protocol is compromised, the supplementary protocol is able to detect it. Each supplementary protocol employs a commitment scheme. The committed value essentially contains all input variables to the message generation function. The supplementary protocol verifies if the input is genuine (via function $\text{ISGEN}()$) and the main protocol is faithfully executed. The supplementary protocols are detailed as follows.

Recall KM-1 algorithm in Section 4.2.1, msg_1 is generated by invoking $msg_1 = f_{msg_1}(S_{pub}, R_{pub})$. Therefore, the publisher as the sender should generate the commitment message

$$com_1 = H(H(S_{pub}) || H(R_{pub}))$$

where $H(x)$ is the secure cryptographic hash function (see Section A.1.2 for background)

and $||$ is the concatenation operator. The publisher sends msg_1 and com_1 to the receiver (the subscriber). The subscriber initiates the supplementary protocol $\mathcal{SP}_{sub}$ as shown in Figure 4.2 to verify the committed value. Upon subscriber's request, the TTP verifies whether or not the uploaded msg_1 and com_1 are truly generated by the publisher. This verification is based on the assumption of communication channel (see Section 3.1.3). If the verification is passed, the TTP signals the publisher to reveal the committed value, i.e., the inputs to f_{msg_1} . Then, the TTP validates if received S'_{pub} is genuine by invoking $ISGEN(S'_{pub})$. Based on S'_{pub} and R'_{pub} , TTP re-calculates the original message msg'_1 and commitment message com'_1 as reference values. They are compared with msg_1 and com_1 which were sent to the subscriber by the publisher. If either the original message or the commitment message is inconsistent or the revealed committed value (i.e., S'_{pub}) is not genuine, TTP will conclude that the publisher has malicious behavior in generating msg_1 and/or com_1 ; otherwise, TTP will prove that the publisher is honest in computing $msg_1 = f_{msg_1}(S_{pub}, R_{pub})$. Finally, the result will be sent to the subscriber. With $\mathcal{SP}_{sub}$, the malicious behavior in performing f_{msg_1} is detectable.

Similarly, msg_2 is generated by invoking $msg_2 = f_{msg_2}(msg_2, S_{sub}, n_1, n_2)$. The subscriber as the message sender should calculate

$$com_2 = H(H(msg_1) || H(S_{sub}) || H(n_1) || H(n_2))$$

The publisher, after receiving msg_2 and com_2 from the subscriber, can initiate the supplementary protocol $\mathcal{SP}_{pub}$ defined in Figure 4.3, to detect whether the subscriber has malicious behavior in calculating f_{msg_2} .

The commitment schemes used in both supplementary protocols employ a hash function to achieve hiding and binding properties. We can prove that by receiving com_1 , the subscriber cannot recover the publisher's private input S_{pub} or random variable set

 Figure 4.3: Protocol of $\mathcal{SP}_{pub}$

R_{pub} . As long as homomorphic encryption is secure and r_i is randomly selected, pre-computation attack [62] is considered as computationally infeasible especially in the context of MMOG where the life cycle of msg_1 is very short. The hiding property is thereby satisfied. Moreover, attempting to construct S'_{pub}, R'_{pub} such that

$$S_{pub} \neq S'_{pub} \vee R_{pub} \neq R'_{pub}$$

$$H(H(S_{pub}) || H(R_{pub})) = H(H(S'_{pub}) || H(R'_{pub}))$$

is equal to find a hash collision. This is also computationally infeasible. Therefore, the binding property is also satisfied. Similar proof can be made for the hiding and binding properties of the commitment message com_2 .

 Figure 4.4: Protocol of $\mathcal{P}_{safe}$

Figure 4.5: Sequence Diagram of SAFE

4.3 SAFE Protocol

With KM-1 and the proposed commitment schemes, we can construct the complete SAFE protocol as follows.

$$\begin{aligned}
 \mathcal{P}_{safe} &::= \langle E, I \rangle \\
 E &= \{pub, sub\} \\
 I &= \{in_{pub}, in_{sub}\} \\
 in_{pub} &= \{pos_{pub}, dr_{pub}, gs_{pub}, insl\} \\
 in_{sub} &= \{pos_{sub}, er_{sub}\}
 \end{aligned}$$

The protocol is performed between the publisher pub and subscriber sub ($E = \{pub, sub\}$). The publisher initially knows its position, DR, game state and the result of SL update since last change ($in_{pub} = \{pos_{pub}, dr_{pub}, gs_{pub}, insl\}$). The subscriber initially knows its position and ER ($in_{sub} = \{pos_{sub}, er_{sub}\}$). The protocol is defined in Figure 4.4 and sequence diagram is illustrated in Figure 4.5. In the SL update stage (step (i) to step (iv)(b)), the subscriber firstly checks if its ER needs to be updated. If so, the subscriber will send $ErUpdate$ to signal the publisher for starting an SL update. The publisher, if receives this message or its position is no longer contained by its DR, will

generate 1) a new DR by invoking $dr_{pub} = \text{GETDR}(pos_{pub})$; 2) a KM-1 message msg_1 by invoking f_{msg_1} ; and 3) a commitment message com_1 . Otherwise, the publisher will skip the SL update stage and start state update stage. The messages msg_1 and com_1 will be sent to the subscriber who executes the verification protocol $\mathcal{SP}_{sub}$. The protocol $\mathcal{P}_{safe}$ will abort if the supplementary protocol detects malicious behavior in the publisher. Otherwise, the subscriber will compute er_{sub} , msg_2 and the corresponding com_2 , and send msg_2 and com_2 to the publisher. The publisher will then execute the supplementary protocol $\mathcal{SP}_{pub}$ to detect if the subscriber has malicious behavior. If it does, the protocol $\mathcal{P}_{safe}$ will abort; otherwise, it will calculate $insl$ from the received message msg_2 . For the state update stage (step (iv)(c) to step (v)(b)), it is identical to the state update stage in protocol $\mathcal{P}_{sp2p}$.

4.4 Summary

In this chapter, we have proposed the SAFE protocol which employs privacy preserving protocol in the SL update to mitigate IE attacks. The SL update protocol in SAFE is a combination of the main protocol $\mathcal{P}_{safe}$ and the supplementary protocols $\mathcal{SP}_{sub}$ and $\mathcal{SP}_{pub}$. The main protocol performs the SL update by privacy preserving protocol. The supplementary protocols further detect the malicious behavior including forged input. In fact, the supplementary protocols are not necessarily executed in every round of $\mathcal{P}_{safe}$. It is because occasionally executing the supplementary protocols is able to detect the malicious behavior with non-zero probability. In this case, the malicious attack is considered as ‘ineffective’ based on the definition in Section 3.3.2 – the attack is effective if and only if it cannot be detected in any cases. The SAFE protocol is a pure software protocol, which does not need any special requirements on hardware, and can be deployed on most platforms. The evaluation of SAFE will be presented in Chapter 6 and Chapter 7. We will find an alternative approach, which will be shown in the next chapter.

Chapter 5

PP2P

In this chapter, we propose an alternative IM scheme to SAFE, which is called Protected P2P (PP2P). Unlike SAFE which mainly relies on the privacy preserving computation to obfuscate sensitive information, PP2P builds a secure place on each peer called trusted block to store and process sensitive information. The idea comes from the observation that in SP2P, IE attacks could happen because HBC or malicious Alice can arbitrarily read or modify sensitive information stored in her (the attacker’s) memory. If we can confine the access to sensitive information to what the protocol defines, IE attacks can be mitigated. PP2P is designed for platforms with advanced security hardware, and thereby allowing more complete protection on sensitive information.

This chapter is organized as follows. Section 5.1 describes the design and possible implementation of the trusted block. Section 5.2 describes the PP2P protocol. The content of this chapter is derived from our published paper ‘Trusted Block as a Service: Towards Sensitive Applications on the Cloud’ [58]. The design and implementation of the trusted block involve technical backgrounds of hardware virtualization and trusted computing which are summarized in Appendix A.2 and Appendix A.3.

Figure 5.1: The trusted block and the legacy system

5.1 Trusted Block

A trusted block is designed as a safe place for storing and processing sensitive information. The terminology *trusted* is defined to describe an entity behaves as what it is designed to [2]. Figure 5.1 illustrates a system on peer's computer where the trusted block is built to perform IM. We can see there is a security boundary between the trusted block and other components which are called legacy system in this work. The boundary achieves *isolation* between the trusted block and the legacy system. The sensitive information is stored in the trusted block as private data. Private functions can be defined in the trusted block to operate private data. Neither private data nor private functions can be directly accessed by the legacy system. Public functions are the only exceptions to override isolation. Those functions are deliberately defined in the trusted block to provide *interaction* interfaces to the legacy system. Public functions are invocable by the legacy system. They allow the legacy system to indirectly interact with sensitive information under the control of what the trusted block defines. The controllable interaction guarantees that sensitive information is always processed in an expected manner. In addition, two trusted blocks on different systems are able to securely exchange data. This involves establishing a *communication* channel between them. The channel should

guarantee security properties such as integrity, confidentiality and authentication. Isolation, interaction and communication are basic functionalities of a trusted block. In the rest of this section, we demonstrate these basic functionalities. We also present an implementation that fulfills these basic functionalities.

5.1.1 Design Rationale

We describe the design rationale of a trusted block, specifically its functionalities such as isolation, interaction and communication.

5.1.1.1 Isolation

Isolation enforces a security boundary between the trusted block and the legacy system. The boundary 1) prevents data and functions in the block from being arbitrarily accessed by the legacy system, and 2) requires the trusted block to be self-contained.

In the trusted block, sensitive information is stored as private data. Only functions defined in the trusted block are allowed to access private data. These functions are classified into ‘private function’ and ‘public function’. Both of them are stored in the trusted block to guarantee they cannot be compromised by the legacy system. Private functions are used to define internal routines so they can be only invoked by other private or public functions in the trusted block. Public functions open invocable interfaces to the legacy system, which will be detailed in the next sub-section. To be clear, private data and functions are prefixed with `_`; whereas public functions are not (e.g., `_privatedata`, `_privateFunction` and `publicFunction`).

On the other hand, isolation requires the trusted block to be self-contained. The execution in the trusted block depends only on components in the trusted block. Hence, the execution of a private or public function in the trusted block should only depend on private data and other private functions.

Figure 5.2: Secure communication

5.1.1.2 Interaction

The trusted block provides interaction with the legacy system by defining public functions. A public function has the same functionality of a private function except that the public function is invocable by the legacy system. Any program in the legacy system can invoke a public function with provided parameters. The public function can process this invocation and return a result to the invoker. Therefore, public functions are the only interface for the legacy system to perform operations that rely on sensitive information. They ensure the data exchange between the trusted block and the invoker is under the control.

However, like a hole on the isolation boundary, public functions may become potential attack surface. For example, the attacker may invoke the public function with sophisticatedly constructed parameters, e.g., forged states. An inappropriately designed trusted block under this attack may lead to information exposure or incorrect functionality. So, the design of public functions should bear this risk and provide necessary precautions on malicious invocations.

5.1.1.3 Communication

In the scenario of IM in DVEs or other distributed systems, it is critical to allow two trusted blocks to securely communicate with each other. This enables sensitive information to be exchanged between peers but still keep them under the protection of trusted

blocks. Because two legacy systems are connected via Internet and there is no direct and physical channel between trusted blocks on different peers' platforms, we have to establish a virtual communication channel for the two trusted blocks. The channel should satisfy the following requirements:

Confidentiality The payload transmitted via the channel can be accessed only by source and destination trusted blocks.

Integrity The destination trusted block can detect if received message is modified.

Authenticity The destination trusted block can verify if received message truly comes from the source trusted block as it claims.

To this end, the communication is performed via packing and unpacking functions, which is illustrated in Figure 5.2. Specifically, the source trusted block *src* invokes packing function

$$msg = _PACK(dst, payload)$$

to 'pack' a piece of information *payload* as a message *msg* for the destination trusted block *dst*. The message essentially contains an encrypted payload with a signature. The encrypted payload is generated by encrypting the payload using the encryption key $pk_{enc_{dst}}$, which guarantees confidentiality. The signature is generated by signing the hash of payload using signing key $sk_{sig_{src}}$. When the secure message reaches the destination trusted block *dst*, the block can invoke unpacking function

$$payload = _UNPACK(src, msg)$$

to recover the payload sent by the source trusted block *src*. The unpacking function firstly recovers the payload from the encrypted payload using the decryption key $sk_{enck_{dst}}$. It will also compute the hash of received payload and will check if the signature is

generated by the source trusted block using the verification key $sigk_{src}$, which achieves authentication and integrity.

The challenge of secure communication is to strictly limit the usage of private keys (i.e., $sk_{sig_{src}}$ and $sk_{enc_{dst}}$). Clearly, we must guarantee only ‘trusted’ blocks can obtain this credential for packing and unpacking. Otherwise, a malicious program that gets access to the private keys would be able to construct a message, claiming it from a ‘trusted’ block and sending it to the destination trusted block without letting its malicious behavior be detected. Moreover, the public keys must be carefully distributed among trusted blocks to prevent man-in-the-middle attack [114]. The implementation must solve these issues to achieve confidentiality, integrity and authenticity properties of the channel.

5.1.2 Implementation

Building a trusted block on an open platform such as PC is challenging. A closed platform such as Xbox [94] (not jailbroken [37]) limits the owner’s operations and guarantees that games running on it cannot be compromised. It is easy to build trusted block on such a closed platform. On the contrary, the owner of an open platform has capabilities of choosing what software to run in the system, including operating system, drivers, libraries and applications. In this case, we cannot guarantee the open platform owner (the player in the game) builds the trusted block in our expected way. However, we can seek mechanisms to verify whether the trusted block is built in an expected way on a remote platform. This will allow a peer to verify whether a message is received from a benign trusted block.

In this thesis, the trusted block is implemented based on hardware virtualization and trusted computing. In the following sub-sections, we detail how isolation, interaction and communication are achieved.

Figure 5.3: Implementation architecture

5.1.2.1 Isolation

The hardware virtualization provides built-in features to achieve isolation. Figure 5.3 illustrates the architecture of this implementation. A middleware called hypervisor [34] is introduced as the manager of the system platform. Hypervisor creates two virtual machines for the legacy system and the trusted block respectively, called legacy VM and trusted VM. The legacy VM includes components that are in the legacy system, i.e., operating system kernel, drivers, libraries and applications. Existing applications shall run without modification. The trusted VM contains only trusted block. Two virtual machines share physical processor resources but are assigned with non-overlapped physical memory. In this case, one VM cannot access another VM's memory and vice versa. The isolation is thereby supported. Note that, it is the hypervisor's responsibility to guarantee memory assigned to two VMs are non-overlapped. Consequently, the integrity of hypervisor is vital to achieve isolation.

5.1.2.2 Interaction

The core of interaction is to allow public functions in a trusted block to be invocable by the legacy system. In our work, the invocation is achieved by *hypercall*, a specific functionality provided by hardware virtualization. Hypercall is a way for VMs to request services provided by the hypervisor. It is implemented as a processor instruction *HYPERCALL*

Figure 5.4: Sequence diagram of public function invocation

and the instruction accepts two operands, a service identifier to indicate which service the VM is requesting and a parameter to the service. We employ hypercalls to bridge the two VMs.

Figure 5.4 illustrates a typical interaction between the legacy system and a trusted block, showing how the legacy system invokes public function $\text{FOOBAR}(param)$ with $param$ as the parameter. It starts by invoking a hypercall with $FOOBAR$ as the service identifier and $param$ as the parameter. Hypervisor receives this hypercall and copies parameter $param$ from the legacy system’s memory space to the trusted block’s memory space so that the trusted block can ‘see’ this parameter. After that, hypervisor locates the corresponding public function $\text{FOOBAR}()$ defined in the trusted block and passes control to it. The trusted block executes $\text{FOOBAR}()$ function and stores return value ret to a predefined memory location. It then invokes a hypercall with specific service identifier $JOBDONE$ to indicate the computation is finished. Upon this hypercall, hypervisor copies ret from the trusted block’s memory space to the legacy system’s memory space. Consequently, hypervisor returns the control to the legacy system, so the legacy system can obtain ret in its memory space and continue its execution. During the interaction, hypervisor acts as the media to transfer legacy system’s invocation to the trusted block, and returns computation result from the trusted block to the legacy system. Similar to

Table 5.1: List of related TPM commands

Command	Definition
$pk_k, kb_k = \text{TPM_CREATEKEY}(type, ref)$	Creates a specific <i>type</i> key pair <i>k</i> bound to TPM and sealed to the PCR with reference value <i>ref</i> . The command returns the public key <i>pk_k</i> and key blob <i>kb_k</i> .
$kh_k = \text{TPM_LOADKEY}(pk_k, kb_k)$	Load key pair <i>k</i> to TPM key slot by providing its public key <i>pk_k</i> and key blob <i>kb_k</i> .
$cert_k = \text{TPM_CERTIFYKEY}(kh_k, kh_{aik})$	Creates a key certificate of key pair <i>k</i> (specified by key handle <i>kh_k</i>), signed by <i>aik</i> (specified by key handle <i>kh_{aik}</i>).
$signature = \text{TPM_SIGN}(kh_k, text)$	Sign <i>text</i> by signing key <i>k</i> (specified by key handle <i>kh_k</i>).
$plaintext = \text{TPM_DECRYPT}(kh_k, ciphertext)$	Decrypt <i>ciphertext</i> by decryption key <i>k</i> (specified by key handle <i>kh_k</i>).

isolation, the functionality of interaction relies on the integrity of hypervisor.

5.1.2.3 Communication

The implementation of communication must guarantee that only the benign trusted block can access to the private keys ($sk_{sig_{src}}$ and $sk_{enc_{dst}}$). Otherwise, the legacy system can pretend itself as a benign trusted block, and send forged messages or obtain information from received messages without being detected. In our work, the requirement is achieved by using trusted computing (see Appendix A.3). Trusted computing is a technology to enhance platform's security, and secure key creation and key distribution are parts of its functionalities.

The technology depends on a secure chip called TPM [126] which provides hardware-based security functionalities. TPM is designed to be tamper-resistant to protect internal data. In particular, Endorsement Key (EK) and Storage Root Key (SRK) are stored persistently in TPM's internal non-volatile memory. They are critical data for TPM. EK is an asymmetric signing key generated by TPM manufacture. The private part of EK is stored in the manufactured TPM. A certificate of EK is shipped with the TPM to endorse that it is a genuine TPM chip. Because EK never leaves TPM, anything signed by EK must be issued from this TPM. SRK is a symmetric encryption key which

is generated on TPM initialization. Similar to EK, SRK never leaves TPM and therefore SRK-encrypted objects can be safely stored outside TPM without being decrypted or compromised. When encrypted objects are loaded into TPM, SRK can decrypt them before checking its integrity. Allowing objects to be stored outside TPM is vital because TPM has limited storage space.

TPM also has volatile memory for Platform Configuration Registers (PCRs) and key slots. The platform configuration is a TPM-specific measurement of how the platform is constructed. Platform Configuration Registers (PCRs) are internal registers protected in TPM's volatile memory for storing these measurements. Measurements are expressed as hash values, which conveys what software is running in the system. TPM has multiple PCRs for measuring different components in the system. For better illustration, we assume TPM has only one PCR (*pcr*) and all measurements use this PCR. This assumption will not change the implementation method of communication.

The measurement starts when the system is going to build the trusted block. First of all, the hypervisor and trusted block should be loaded into memory. A specific instruction called Late Launch [91] is then invoked with operands including the memory region that contains the hypervisor and trusted block, and the entry point of the hypervisor initializer code. This invocation is called Dynamic Root of Trust Measurement (DRTM) [70] which is a standard method to initialize measurement by TPM. The instruction guarantees the following conditions are satisfied atomically: 1) the PCR is set to the hash value of content in specified memory region; 2) the system starts to execute the code specified by the entry point; and 3) no other code is being executed. The first condition performs the measurement of the hypervisor and trusted block. If the loaded components are the benign hypervisor and trusted block, the PCR must be a unique value (e.g., *pcr* = *benign_tb*), representing the trusted block is built in our expected way. The second and third conditions guarantee the entry point of the hypervisor (i.e., initializer code) is the

only code being executed in the system. In other words, this initializer code takes full control of the system and therefore the behavior of the system is completely measured by the PCR. The initializer code performs essential operations such creating two VMs containing the legacy system and trusted block, configuring isolation between them, registering hypercall handlers and so on. After that, it passes the control to the legacy system. We stress that if either the hypervisor or trusted block is compromised, the PCR will be a different value than *benign_tb*. Therefore, the PCR values can tell whether the genuine trusted block or hypervisor has been compromised.

Key slots are temporary storage for loading keys in the TPM volatile memory. If a key is loaded to a key slot, TPM will assign a key handle for further operations such as signing/verification and encryption/decryption. Particularly, Attestation Identity Keys (AIKs) are signing keys that can be loaded in the slots for authenticating whether or not a piece of information is from this TPM. We assume every TPM on each peer has generated an AIK and its public part is known by TTP. If TTP receives a message signed by an AIK, the TTP will know this message is generated by particular TPM.

The trusted block can interact with TPM via TPM commands. Related TPM commands in our work and their usages are summarized in Table 5.1. TPM provides rich features for key creation, which is implemented by TPM command `TPM_CREATEKEY(type, ref)`. The invoker must specify the type of key pair, which can be *ENC* for encryption/decryption or *SIG* for signing/verification. The command returns a public key pk_k and a key blob kb_k . The key blob contains private key sk_k encrypted by SRK, making it only accessible to the TPM that creates the key. This is called *key binding*, which protects the private key from being arbitrarily used by entities other than this TPM. Recall that secure communication requires that private keys $sk_{sig_{src}}$ and $sk_{enc_{dst}}$ should be only accessible to the genuine, uncompromised trusted blocks. Therefore, binding the keys to a TPM is not enough for achieving this requirement. Fortunately, TPM provides

another functionality called *key sealing*, which seals the generated private key to a specified platform configuration by associating it to specific PCR value. That is, a key is only accessible to the trusted block whose platform status is represented by the PCR value. Technically, key sealing is achieved by specifying parameter *ref* as the reference PCR value in the TPM key creation command.

After a key pair is created with binding and sealing, we must securely distribute the public key to other peers. In detail, the distribution must allow the receiver to believe 1) the key pair is created by a genuine TPM; 2) the private key is bound to the TPM; and 3) the private key is sealed to the reference PCR value (i.e., *beign_tb*) that represents the trusted block is built in our expected way. Otherwise, one can create a key pair by software and send its public key to the receiver, without any limitations on using its private key. TPM provides a command `TPM_CERTIFYKEY(kh_k, kh_aik)` for secure key distribution. The command issues a key certificate *cert_k* for the key pair *k* where *kh_k* and *kh_aik* are the key handles of *k* and AIK respectively. A key handle is obtained by loading the key to TPM's key slot, which can be achieved by command `TPM_LOADKEY(pk_k, kb_k)` where *pk_k* and *kb_k* are the public key and key blob respectively. The certificate includes the public key and reference PCR value, and is signed by TPM's AIK. If the receiver obtains this certificate, he/she can verify whether the key pair (whose public key is claimed in the certificate) is generated by a genuine TPM and sealed to a genuine trusted block.

In our work, we rely on TTP to verify key certificates and distribute public keys to peers. Figure 5.5 illustrates the sequence of key creation, verification and distribution. The trusted block *x* interacts with TPM by issuing commands. The AIK is firstly loaded into TPM for obtaining its key handle *kh_aik_x*. The trusted block creates encryption and signing key pairs respectively. For each key pair, the private key is bound to the TPM and is sealed to the hash value that represents good status of the trusted block and

Figure 5.5: Sequence diagram of key generation, verification and distribution

hypervisor($benign_tb$). The key is loaded into key slot and key handle is obtained. With this key handle and AIK handle, key certificate can be generated. Two key certificates are transmitted to TTP for verification. If passed, TTP will generate a message which contains all public keys, and distribute the message to all peers.

TPM keys are used in packing and unpacking functions. In the packing function `_PACK()`, source trusted block's private signing key sk_sig_{src} must be employed to sign the message. When the key is created by the TPM with binding and sealing properties, the private key is not revealed to the trusted block. Instead, the trusted block only receives the public key and key blob. The trusted block obtains key handle kh_sig from the TPM using command `TPM_LOADKEY( $pk\_sig_x, kb\_sig_x$ )` and requests the TPM to sign the message using command `TPM_SIGN( $kh\_sig, text$ )`, where $text$ is the data to be signed. Especially, the signing command inspects the sealing information of loaded signing key by comparing current PCR value (pcr) to the reference PCR value specified in the key creation command (i.e., ref in key creation command `TPM_CREATEKEY( $type, ref$ )`). In case they are mismatched, the signing command will return an error. Similarly, in the unpacking function, destination trusted block requires private encryption key sk_enc_{dst} to decrypt the message in order to recover payload. To do this, destination trusted block uses key handle kh_enck to invoke command `TPM_DECRYPT( $kh\_enc, ciphertext$ )` to decrypt $ciphertext$. Similar to command `TPM_SIGN( $kh\_sig, text$ )`, TPM also checks sealing information of the loaded encryption key.

5.2 PP2P Protocol

The PP2P protocol is derived from SP2P but relies on trusted blocks for protecting sensitive information and providing necessary operations. For each peer x , the trusted block and the IM logic in the legacy system can be modeled as two entities. The sensitive information and operations are protected in the trusted block, and this entity is called *trusted IM x_trust* . The rest of IM is called *legacy IM x_legacy* and is executed in the legacy system. The legacy IM deals with insensitive operations and offloads operations on sensitive information to the trusted IM via invoking specific public functions.

The PP2P protocol is defined as follows.

$$\begin{aligned}
\mathcal{P}_{pp2p} &::= \langle E, I \rangle \\
E &= \{pub_legacy, pub_trust, sub_legacy, sub_trust\} \\
I &= \{in_{pub_legacy}, in_{pub_trust}, in_{sub_legacy}, in_{sub_trust}\} \\
in_{pub_legacy} &= \{pos_{pub}, gs_{pub}\} \\
in_{pub_trust} &= \{-cer_{sub}\} \\
in_{sub_legacy} &= \{pos_{sub}, er_{sub}\} \\
in_{sub_trust} &= \emptyset
\end{aligned}$$

The protocol is performed among the legacy and trusted IMs on both publisher and subscriber ($E = \{pub_legacy, pub_trust, sub_legacy, sub_trust\}$). The publisher's legacy IM initially knows its position and game state ($in_{pub_legacy} = \{pos_{pub}, gs_{pub}\}$) and the publisher's trusted IM initially knows the subscriber's ER since last change ($in_{pub_trust} = \{-cer_{sub}\}$). The subscriber's legacy IM initially knows its position and ER ($in_{sub_legacy} = \{pos_{sub}, er_{sub}\}$), and the subscriber's trusted IM initially knows nothing. In the rest of this section, we illustrate the trusted IM and legacy IM respectively.

5.2.1 Trusted IM

Figure 5.6 and Figure 5.7 illustrate the behavior of entities *pub_trust* and *sub_trust* respectively. The trusted IM keeps sensitive information like the subscriber's ER and the publisher's game state under protection. Operations required by the legacy IM on these private data are defined as public functions.

The ER update message is generated by public function $GENERMSG(pub, er)$ where *pub* refers to the destination and *er* is the subscriber's ER. This function packs the ER as the message payload and returns the message that can be safely transmitted to the publisher's trusted IM. On the side of the publisher, the trusted IM provides a public function $CACHEER(sub, ermsg)$ which accepts *sub* as the source trusted block and *ermsg* as the ER message. This function unpacks received subscriber's ER update message to recover ER payload. The ER, if it passes the genuine test, will be cached as private data *_cer_{sub}*. Another public function $INSL(sub, pos)$ is defined in the publisher's trusted

```
1: /* Publisher's trusted IM */
2: /* Private data:  $\_cer_{sub}$  */
3: /* Private functions: */
4:  $\_PACK(dst, payload)$ ,  $\_UNPACK(src, msg)$ ,  $\_ISGEN(imst)$ 
5: /* Public functions */
6: function CACHEER( $sub$ ,  $errmsg$ )
7:    $er \leftarrow \_UNPACK(sub, errmsg)$ 
8:   if  $\_ISGEN(er)$  then
9:      $\_cer_{sub} \leftarrow er$ 
10:    return Success
11:   else
12:    return SubCheat
13:   end if
14: end function
15: function INSL( $sub$ ,  $pos$ )
16:   if  $\_ISGEN(pos)$  then
17:    return  $pos \in \_cer_{sub}$ 
18:   else
19:    return PubCheat
20:   end if
21: end function
22: function GENGSMMSG( $sub$ ,  $gs$ )
23:    $gsmsg \leftarrow \_PACK(sub, gs)$ 
24:   return  $gsmsg$ 
25: end function
```

Figure 5.6: The publisher's trusted IM (pub_trust)

IM to reveal $insl$. The function accepts sub as the source trusted block and pos as the publisher's position. Note that, the parameter pos will be checked if it is a genuine position before it is compared with the cached ER ($_cer_{sub}$). This design prevents the legacy IM from invoking the function with forged position to probe the cached ER. Similar to the ER update message, the game state update message gs_{pub} is generated by public function $GENGSMMSG(sub, gs)$ where sub is the destination trusted block and gs is the game state provided by the legacy IM. This game state update message can be safely sent to the subscriber's trusted IM. Public function $REVEALGS(pub, msg, aoi)$ is defined in the subscriber's trusted IM in order to recover the publisher's game state and decide

```
1: /* Subscriber's trusted IM */
2: /* Private functions: */
3:  $\_PACK(dst, payload)$ ,  $\_UNPACK(src, msg)$ ,  $\_ISGEN(imst)$ ,  $\_GETPOS(gs)$ 
4: /* Public functions */
5: function GENERMSG(pub, er)
6:   msg  $\leftarrow$   $\_PACK(pub, er)$ 
7:   return msg
8: end function
9: function REVEALGS(pub, gsmsg, aoi)
10:  gs  $\leftarrow$   $\_UNPACK(pub, gsmsg)$ 
11:  if  $\neg \_ISGEN(aoi)$  then
12:    return SubCheat
13:  else if gs = Error  $\vee \neg \_ISGEN(gs)$  then
14:    return PubCheat
15:  else
16:    pospub  $\leftarrow$   $\_GETPOS(gs)$ 
17:    if pospub  $\in$  aoi then
18:      return gs
19:    else
20:      return Ud
21:    end if
22:  end if
23: end function
```

Figure 5.7: The subscriber's trusted IM (*sub_{trust}*)

whether it can be revealed to the subscriber's legacy IM. The subscriber's AOI *aoi* is provided by the subscriber's legacy IM and thereby it must be checked if it is genuine. The publisher's game state *gs* must also be checked because it is recovered from received message. If verification is passed, the function will extract the publisher's position from the game state and compare it with the subscriber's AOI. If the position is inside the AOI, the game state must be relevant and will be returned to the invoker; otherwise, the function will return *Ud*.

Figure 5.8: Legacy IM (*pub_legacy* and *sub_legacy*)

5.2.2 Legacy IM

Figure 5.8 illustrates the behavior of legacy IM. The subscriber computes its AOI (aoi_{sub}) (step (i)(a)) and compares the AOI with the ER (step (i)(b)). When the existing ER

does not cover the AOI, the ER update starts. The ER update message msg_{er} is generated by invoking public function $msg_{er} = \text{GENERMSG}(pub, er_{sub})$ (step (i)(b)(2)). If this ER update message is received (step (ii)(a)), the publisher's legacy IM will invoke public function $\text{CACHEER}(sub, msg_{er})$ to cache the received ER (step (ii)(a)(1)). Then, the publisher invokes public function $insl = \text{INSL}(sub, pos_{pub})$ to update the subscriber list (step (ii)(b)). If $insl = true$, the game state update message will be prepared via invoking public function $msg_{gs} = \text{GENGSMSG}(sub, gs_{pub})$ (step (ii)(c)). The subscriber, after receiving game state update message, will invoke public function $ret = \text{REVEALGS}(pub, msg_{gs}, aoisub)$ to reveal the publisher's game state (step (iii)(a)(1)). The result ret can be either gs_{pub} if it is relevant or Ud if otherwise. During the execution, the protocol should abort if the trusted function indicates cheating behavior (returns *PubCheat* or *SubCheat*).

5.3 Summary

In this chapter, we have proposed an IM scheme called PP2P. PP2P protocol is derived from SP2P protocol but employs trusted blocks to process sensitive information. The trusted blocks provide isolation from the legacy system while defining public functions for interaction. Moreover, the trusted blocks can securely exchange data via established secure communication channel. By employing the trusted blocks, PP2P protocol is constructed. The protocol splits the IM into trusted and legacy entities. Trusted IM deals with sensitive information and legacy IM is responsible for other insensitive tasks. In fact, the boundary between the trusted IM and legacy IM can be adjusted. It is possible to move more IM functions into the trusted IM, but doing this may not necessarily improve the mitigation to IE attacks. Trusted blocks must be self-contained for isolation and complicated dependencies such as the handling network protocols are difficult to be ported to the execution environment of the trusted blocks. Therefore, PP2P currently

picks the only necessary components to the trusted IM, leaving all others to the legacy IM.

The PP2P is designed to mitigate IE attacks in both SL update and game state update stages. So far, we have proposed two IM schemes based on different methodologies. In the next chapter, we will evaluate the existing and newly proposed IM schemes by formal methods, in order to verify their effects to mitigate IE attacks.

Chapter 6

Formal Verification

In this chapter, we apply formal verification to verify two vital properties: correctness and secrecy-1. Formal verification is a technique to prove or disprove the correctness of certain properties using formal methods [32]. Generally, there are two well-established approaches to formal verification. Model checking [31, 39] is the approach that relies on systematically exhaustive exploration of state space. Another approach is theorem proving [100] which is the mechanization of a logical proof. In this thesis, model checking is selected as our formal verification technique to complete the task. The essence of model checking involves 1) modeling the system being verified, 2) specifying properties into specifications, and 3) running the tests to verify whether specified properties are satisfied or not. Particularly, if a property is unsatisfied, detailed counter example will be given to explain how the property is unsatisfied.

We start the system modeling by abstracting IM-related states, followed by modeling the behavior of the lite game with specific IM protocol in CSP#. We also model the adversary behavior of the attacker and combine it into the system model. Then, we specify the correctness and secrecy-1 properties as reachability tests. Four IM schemes, including the existing CS and SP2P schemes and our newly proposed SAFE and PP2P protocols, are automatically verified. The verification results will be given in detail.

Figure 6.1: Model Checking

6.1 System Modeling

In this section, we firstly illustrate the rationale of model checking and the syntax of CSP# language. We then discuss how the system model is generated, which is the very first step in the process of model checking. It involves modeling the lite game, providing necessary abstraction of IM-related states, and converting IM protocols into modeling language. Finally, the adversary behavior of IE attacks is also modeled and combined into the system model. The complete verification code can be found in Appendix B.

6.1.1 Model Checking and CSP# Language

The process of model checking can be summarized in Figure 6.1. System model and property specifications are two inputs to the model checker. The model is generated by modeling the lite game and the adversary behavior. The specifications are generated by specifying correctness and secrecy-1 properties. When the system model and property specifications are ready, formal verification can be automatically performed by the model checker. Each property will be verified. Especially, if a property is invalid, counter examples will be provided by the model checker.

In our work, we adopt Process Analysis Toolkit (PAT) [1] as the model checker. PAT depends on CSP# language for modeling and property specification. CSP# extends

Hoare’s CSP [63, 64] with new language features. It integrates high-level modeling operators (e.g., parallel composition, choice, interrupt, channel communication, etc.) with shared variable and low-level procedural codes, for the purpose of efficient mechanical system verification. In particular, CSP# supports user-defined data types via external C# library. Part of CSP# syntax, which have been used in this chapter, is given below:

$Process\ P ::= Stop$	– termination
$ e \rightarrow P$	– event prefixing
$ e\{program\} \rightarrow P$	– data operation prefixing
$ [b]P$	– state guard
$ c!m \rightarrow P$	– channel output
$ c?m \rightarrow P(m)$	– channel input
$ P; Q$	– sequential composition
$ ifa\ (b)\ \{P\}\ else\ \{Q\}$	– conditional branch
$ P \square Q$	– external choices
$ P Q$	– interleaving composition
$ P \triangle (e \rightarrow Q);$	– interrupt

The fundamental of CSP# is concurrency and communication. In its syntax definition, P and Q are processes, e is an event, b is a Boolean expression and c is a channel. A process can be used to model a series of behaviors of an entity in the protocol. An event is a basic unit of process, which is used to model state transitions. $program$ in ‘ $e\{program\} \rightarrow P$ ’ is data operation (i.e., a piece of code to change states) and is executed atomically with the event e . The data operation is optional. In ‘ $[b]P$ ’, process P can be executed only if Boolean expression b is *true*. Channel can be used to model information exchange between two processes. The channel output event $c!m \rightarrow P$ is used to send message m to channel c , and the channel input event $c?m \rightarrow P(m)$ is used to receive message m from the channel (and this message m is accessible to P). The message m can be a list of variables separated by ‘.’ (e.g., $c!param1.param2$) and the receiver should receive this message accordingly (e.g., $c?param1.param2$). In this thesis, we model the communications between entities by using synchronous channels (channels with zero buffer size). The synchronous channel guarantees that the channel output event (which

models message sending) and the channel input event (which models message receiving) must execute at the same time. A combined event $c.m$ will be generated in the event trace. The sequential composition ' $P; Q$ ' is used to model the behavior that ' P is executed and then Q is executed'. It can be used to model two sequential steps in the protocol, e.g., the SL update and game state update stages. The conditional branch ' $\text{ifa } (b) \{P\} \text{ else } \{Q\}$ ' is used to model the behavior that ' P is executed if $b = \text{true}$; otherwise Q is executed'. Note that, given b , execution path (P or Q) is determined when the conditional branch is executed. It can be used to model branches in the protocol. The external choice ' $P \sqcap Q$ ' is used to model the behavior that 'either P or Q is executed'. Compared to the conditional branch, the behavior of external choice is nondeterministic (the next process being executed can be either P or Q). The behaviors of both P and Q should be explored by the model checker. The external choice is commonly used to model nondeterministic states (e.g., whether AOI is within ER) and attack behaviors (e.g., sending benign or forging ER update message). The interleaving composition ' $P \parallel Q$ ' allows processes P and Q to execute independently unless they wait for each other for communications via synchronous channels. It can be used to model independent behaviors of different entities such as publisher and subscriber. The interrupt ' $P \triangle (e \rightarrow Q)$ ' behaves as P until event e is engaged and then behaves as Q . It can be used with synchronous channels to model the behavior like 'if game state update message msg_{pub} is received, the protocol outputs msg_{gs} (modeled by P); if msg_{gs} is not received (modeled by e), the protocol will output Ud (modeled by Q)'.

A CSP# program can be translated into a Labeled Transition System (LTS) [121] which provides the basis for automatic system analysis and verification. An LTS is a 3-tuple $(S, \text{init}, \rightarrow)$ that consists of all possible states S , initial system configuration $\text{init} \in S$, and a set of labeled transition relations $\rightarrow$. A state is represented by (V, P) , where V refers to a valuation function mapping a variable name to its value, P is the

```

var e_output;

LiteGame() =
  Move();
  aliceispub → Update_Expected_Output(ALICE, BOB);
  Im(ALICE, BOB);
  bobispub → Update_Expected_Output(BOB, ALICE);
  Im(BOB, ALICE);
  LiteGame();

Im(pub, sub) = ...;

```

Figure 6.2: Modeling the lite game

current process expression. A transition is thereby defined as a labeled directed relation from a source state to a target state. Among various property tests, *reachability* test is used most in this thesis to verify if certain property is satisfied. The test checks if a specified state is reachable. This is defined by if there exist transitions from initial system configuration *init* to the state specified by the property.

6.1.2 Modeling the Lite Game

The lite game is used to test IM schemes. We model the behavior of the lite game as a CSP# process *LiteGame*, which is shown in Figure 6.2. The first behavior is that peers move in the game, which is modeled as process *Move()*. After that, Alice acts as the publisher and Bob acts as the subscriber to perform a specific IM scheme. Event *aliceispub* is defined as a marker for providing better readability in event trace. When *aliceispub* is found in the event trace, we know the following events are generated when Alice acts as the publisher and Bob acts as the subscriber. Process *Update_Expected_Output(ALICE, BOB)* calculates the expected output of the IM protocol (i.e., the result of f_{im}). The expected result will be save as *e_output* (defined in Figure 6.2). This variable will be used to testify the correctness and secrecy-1 properties. The process *Im(ALICE, BOB)* models the behavior of the IM protocol with Alice as the publisher and Bob as the subscriber.

The definition of Im is different for various IM schemes. Subsequently, Alice and Bob swap their roles in IM. This is marked by event *bobispub*. The expected output e_output will be updated by $Update_Expected_Output(BOB, ALICE)$. The IM protocol will be executed with Alice as the subscriber and Bob as the publisher. This is modeled by $Im(BOB, ALICE)$. At this time, the game finishes the current round and will continue to the next round. This is modeled by executing itself.

6.1.3 Abstraction of IM-related States

The core of system modeling is to model the behavior of a system by exploring its states and their transitions. IM protocols involve operations on IM-related states such as one's position, AOI, ER, DR and game states. In a real game, these states are normally expressed as coordinates or explicit values. If we model these IM-related states by explicit values, the number of system states are excessive. Exploring these excessive states is infeasible and unnecessary for model checking. We need to seek an approach to abstract these IM-related states, to reduce system states while keeping necessary details for property specifications.

Further investigation on IM protocols shows that the behavior of the system depends only on the relationships between states rather than the explicit values of the states. Additionally, correctness and secrecy-1 properties can be specified without knowing the explicit values of the states. That is, as long as we know the relationship, in order to determine the behavior of the system and specify the properties, it is unnecessary to know the explicit values of the states. Therefore, we can focus on the relationships between states rather than the states themselves. The relationship between two states can be defined as a function, called *relationship function*. The function essentially includes the semantics of each state (e.g., 'the subscriber's AOI' and 'the subscriber's ER') as inputs, the relationship definition (e.g., whether 'the subscriber's AOI' is contained with 'the

Table 6.1: Symbolic representations of IM-related states

State	Representation
$Pos(x[, o_insl])$	Peer x 's position
$Aoi(x[, o_rel])$	Peer x 's AOI
$Er(x[, o_insl])$	Peer x 's ER
$Dr(x[, o_insl])$	Peer x 's DR
$Gs(x[, o_rel])$	Peer x 's game state

Table 6.2: Relationship functions

Benign result	Function	Definition	Scheme
<i>aoiiner</i>	$ISAOIINER(aoi, er)$	$aoi \subset er$	SP2P, SAFE, PP2P
<i>posindr</i>	$ISPOSINDR(pos, dr)$	$pos \in dr$	SAFE
<i>insl</i>	$ISINSL(pos, er)$	$pos \in er$	SP2P, PP2P
	$ISINSL(dr, er)$	$dr \cap er \neq \emptyset$	SAFE
<i>rel</i>	$ISREL(gs, aoi)$	$GETPOS(gs) \in aoi$	All

subscriber's ER') and the result of this function (e.g., *true* or *false*). Since the inputs of the function focus only on the semantics rather than the values of the states, the states can be modeled as symbols. A symbol should indicate the owner of the state and whether the state is genuine. The relationship definition describes the functionality. The result is a Boolean value, representing the calculation result when the relationship function is evaluated with two genuine states as the inputs. When a state is updated, its symbol remains the same. However, the results of functions that require the state as input must be updated accordingly to reflect the change.

Table 6.1 summarizes all IM-related states and their symbolic representations. Each type of state is modeled as a symbol with ownership as the identifier and optionally an overridden value prefix with *o_*. This overridden value can be used to model forged states, which will be detailed soon later. The value of x can be either publisher *pub* or subscriber *sub*.

Table 6.2 summarizes all relationship functions involved in this thesis. Each function accepts two IM-related states as the parameters in column 'Function'. The definitions of the functions are given in column 'Definition'. When parameters are benign states, the

results of the functions are also benign. These results are denoted as Boolean variables in column ‘Benign result’. Function $\text{isAOIInER}(aoi, er)$ defines the belonging relationship between the subscriber’s AOI aoi and ER er , which is used in ER update in SP2P, SAFE and PP2P protocols. Function $\text{isPosInDR}(pos, dr)$ defines the membership relationship between the publisher’s position pos and DR dr , which is used in DR update in SAFE protocol. Function $\text{isInSL}()$ models the result of SL update and has two versions. In SP2P and PP2P protocols, the function is defined as the membership relationship between the publisher’s position pos and subscriber’s ER er . In SAFE protocol, the function is defined as the disjointness relationship between the publisher’s DR dr and subscriber’s ER er . Function $\text{isREL}(gs, aoi)$ defines the membership relationship between the publisher’s position (extracted from the game state gs) and the subscriber’s AOI aoi , which is used in game state update in all protocols.

We give an example to explain how relationship functions work here. In SP2P protocol, the ER update requires the subscriber to compare its AOI aoi_{sub} with ER er_{sub} , to determine whether or not the ER needs an update. If the AOI and ER are modeled by explicit values, their states can be excessive. However, whether or not updating the ER is determined by the belonging relationship between the AOI and ER (i.e., the result of $aoi_{sub} \subset er_{sub}$) rather than the individual values of the AOI and ER. In this case, $Aoi(sub)$ and $Er(sub)$ symbolically model the subscriber’s AOI and ER respectively. Function $\text{isAOIInER}(aoi, er)$ models the relationship between a symbolic AOI (aoi) and a symbolic ER (er). A variable called $aoiiner$ is defined to model the result of this function, which is $aoi \subset er$. When the subscriber’s AOI is changed (e.g., caused by the movement in the VE), the symbol of the subscriber’s AOI is kept as $Aoi(sub)$. Instead, variable $aoiiner$ that models the relationship between AOI and ER must be updated for a new value that represents the relationship between the updated AOI and ER.

The new $aoiiner$ can be either *true* or *false*. As shown in Figure 6.3, process $Det_AoiInEr$ models this nondeterministic state of $aoiiner$. Events $subaoi_in_suber$ and

$$\begin{aligned}
 Det_AoiInEr(pub, sub) = & \\
 & subaoi_in_suber\{aoiiner = true; \} \rightarrow Skip \square \\
 & subaoi_notin_suber\{aoiiner = false; \} \rightarrow Skip;
 \end{aligned}$$

 Figure 6.3: Modeling the nondeterministic state of *aoiiner*

$$\begin{aligned}
 Det_InSl(pub, sub) = & \\
 & sub_in_pubsl\{insl = true; \} \rightarrow Skip \square \\
 & [!rel]sub_notin_pubsl\{insl = false; \} \rightarrow Skip;
 \end{aligned}$$

 Figure 6.4: Modeling the nondeterministic state of *insl*

subaoi_notin_suber model the situations when *aoiiner* is *true* and *false* respectively. Updating *aoiiner* is modeled as data operations of these two events. An external choice connects these two events to model the nondeterminism.

Similarly, updating *insl* can be modeled by process *Det_InSl* as shown in Figure 6.4. Events *sub_in_pubsl* and *sub_notin_pubsl* model the situations when *insl* is *true* and *false* respectively. In particular, if the publisher's game state is relevant to the subscriber (i.e., *rel = true*), the subscriber must be on the publisher's SL (i.e., *insl = true*). Therefore, it is only possible to set *insl = false* when *rel = false*. This is modeled as a guard condition *[!rel]* of event *sub_notin_pubsl*.

6.1.4 Behavior of IM Schemes

When an IM scheme is faithfully executed, its behavior is defined by the IM protocol itself. Therefore, modeling the benign behavior of an IM scheme is to convert its protocol into CSP# statements.

A protocol essentially consists of steps to be performed by peers. Each step can be either a local computation or communication with another peer. The local computation can be modeled as a CSP# event with data operations responsible for state updating. For communications, they can be modeled by sending or receiving data via synchronous

```

channel sub2pub 0;
channel noermmsg 0;

SP2P_Sub_Benign_SlUpdate(pub, sub) =
  ifa (!ISAOIINER(Aoi(sub), Er(sub))) {
    updateer { aoiiner = true; } →
    Det_InSl(pub, sub);
    sub2pub!Er(sub) → Skip
  } else {
    noermmsg!0 → Skip
  };

SP2P_Sub_Benign_GsUpdate(pub, sub) =
  (
    pub2sub?gsmmsg →
    ifa (ISGEN(gsmmsg) == false) {
      cheat → Stop
    } else {
      protocol_output {
        var rel = ISREL(gsmmsg, Aoi(sub));
        if (rel) { a_output = gsmmsg; } else { a_output = Ud(); }
        output_ready = true;
      } → Skip
    }
  ) interrupt (nogsmmsg?0 → Skip);
    
```

Figure 6.5: An example of modeling benign subscriber in $\mathcal{P}_{sp2p}$

channels. Data in the channel is transferred from the sender to the receiver. Some steps in the protocol are executed sequentially. In this case, their corresponding CSP# statements are also modeled as sequential events or sequential composition of processes (e.g., the SL update and game state update stages). Branches in the protocol can be modeled by conditional choices, i.e., $ifa(b)\{P\}else\{Q\}$ where e is the condition, P is the *true* branch and Q is the *false* branch. To not be confused with the external choice, the conditional choice let the model checker explore only one branch determined by b .

For example, Figure 6.5 models benign behavior of the subscriber in SP2P. The behavior of the SL update is modeled by process $SP2P_Sub_Benign_SlUpdate(pub, sub)$.

The conditional choice checks if the subscriber's AOI is inside its ER (*aoiiner*). If not, the subscriber will update its ER to cover AOI. This is modeled as a data operation *aoiiner = true* of event *updateer*. Since ER is updated, the relationship between ER and the publisher's position might be changed as well. This affects the value of *insl*. In this case, process *Det_InSl(pub, sub)* is invoked to model this uncertainty (*insl* can be either *true* or *false* after ER is updated). The event *updateer* and process *Det_InSl(pub, sub)* are modeled in process *SP2P_Sub_Benign_SlUpdate(pub, sub)* to keep *aoiiner* and *insl* updated. After that, the ER update message *Er(sub)* is sent to the publisher, which is modeled as sending the subscriber's ER via synchronous channel *sub2pub* (*sub2pub!Er(sub)*). The receiver, the publisher in the IM scheme, should also wait on this channel for receiving ER updates. Alternatively, if the subscriber's AOI is inside its ER (*aoiiner = true*), synchronous channel *noerrmsg* will be signaled so the publisher knows there is no ER update message.

The subscriber's behavior of the game state update is modeled by process *SP2P_Sub_Benign_GsUpdate(pub, sub)*. If game state message *gsmg* is received via channel *pub2sub* (*pub2sub?gsmg*), the subscriber verifies if the received game state is genuine via invoking *ISGEN(gsmg)*. If the state is not genuine, the subscriber considers it as an attack and aborts the execution (*cheat* $\rightarrow$ *Stop*). If the state is genuine, the subscriber calculates whether or not the received game state is relevant and generates corresponding protocol output, which is modeled as data operation of event *protocol_output*. If game state message *gsmg* is not received, the protocol outputs *Ud*. The protocol output is saved to a variable called *a_output* (a.k.a., actual output) and *output_ready = true* is set to indicate the protocol output is ready. These variables will be used to testify correctness and secrecy-1 properties.

```

...
sub2pub?ermesg{
  if (beh == HBC) {k.collect(ermesg); }
} →
...

```

Figure 6.6: An example of modeling HBC behavior

6.1.5 Adversary Behavior

Along with benign behavior of the protocol, attacking behavior is also modeled as a part of system model. To classify the behavior of peer (e.g., Alice), variable *beh* is introduced. Its value can be *BEN* (short for benign), *HBC* (short for curious-but-honest) or *MAL* (short for malicious).

HBC Alice faithfully executes the protocol but may remember the information that passes to her, such as the messages or intermediate results, and try to learn additional information as much as possible from what she has obtained. According to this definition of HBC behavior, we can reuse the model of benign Alice but additionally model her capability of collecting messages of intermediate results and deducing information. In particular, we introduce variable *k* to model Alice’s knowledge. Function *k.collect(info)* can be invoked in an event’s data operation to collect *info* as her knowledge. Function *k.know(info)* is defined to return a Boolean value to indicate whether *info* can be deduced from Alice’s knowledge. For instance, if Alice collects a message (*k.collect(msg)*), she can then deduce the hash value of the message (*k.know(H(msg)) = true*). Figure 6.6 shows a code snippet of modeling HBC behavior. The publisher receives ER update message *ermesg*. If the publisher is benign, nothing additional will happen; however, if the publisher is HBC Alice, she will add this message to her knowledge.

For malicious Alice, we focus on modeling her observable behavior like sending messages. The local behavior is not affecting the attacker’s knowledge, because it is not

```

...
sub2pub!Er(sub) → Skip □
noerrmsg!0 → Skip □
sub2pub!Er(sub, true) → Skip □
sub2pub!Er(sub, false) → Skip
...

```

Figure 6.7: An example of modeling the subscriber’s malicious behavior on SL update

$$\begin{aligned}
 \text{ISINSL}(pos, er) &= \begin{cases} pos.o_insl & \text{if } pos \text{ is forged} \\ er.o_insl & \text{if } er \text{ is forged} \\ insl & \text{otherwise} \end{cases} \\
 \text{ISINSL}(dr, er) &= \begin{cases} dr.o_insl & \text{if } dr \text{ is forged} \\ er.o_insl & \text{if } er \text{ is forged} \\ insl & \text{otherwise} \end{cases} \\
 \text{ISREL}(gs, aoi) &= \begin{cases} gs.o_rel & \text{if } state \text{ is forged} \\ aoi.o_rel & \text{if } aoi \text{ is forged} \\ rel & \text{otherwise} \end{cases}
 \end{aligned}$$

Figure 6.8: Modeling overridden results of relationship functions

```

...
updatesl{ observed_insl = ISINSL(Pos(pub), cer); } → Skip;
...

```

Figure 6.9: Modeling of publisher’s evaluation of *insl*

observable by the other peer, is not necessary to be modeled. In this case, we can enumerate every possible message that can be sent by using external choices. Let us take the subscriber’s malicious behavior on SL update as an example (see Figure 6.7). In the IM protocol, the subscriber should send its ER update message to the publisher if and only if the ER needs to be updated. However, for malicious Alice as the subscriber, she may not follow the protocol. She may force her ER to be updated and sent to the publisher; she may skip the ER update by signaling *noerrmsg* channel; or she may construct a forged ER $Er(sub, o_insl = true)$ or $Er(sub, o_insl = false)$.

When a forged state is received by benign Bob, it may affect the result of relationship

functions. Forged position, ER, and DR may affect function $\text{ISINSL}()$ and forged AOI and game state may affect function $\text{ISREL}()$ (see Figure 6.8). The result of relationship function will be changed to the overridden value specified when the state is forged. Note that the overridden result of function $\text{ISINSL}()$ has two definitions for different IM schemes (SP2P/PP2P and SAFE). For functions ISAOIINER and ISPOSINDR , their inputs are local states provided by the invoker. When the invoker is benign (e.g., Bob or benign Alice), the local inputs must be benign. When the invoker is malicious (e.g., malicious Alice), the invoker can modify the protocol, directly performing or skipping the ER/DR update without the need to evaluate the relationship function. Therefore, it is not necessary to model the results of these functions when parameters are forged states.

Let us take the publisher's benign behavior on SL update as an example to illustrate how a forged ER affects the evaluation of $\text{ISINSL}()$ (see Figure 6.9). The publisher updates its SL and saves the result to *observed_insl*. Essentially, it invokes $\text{ISINSL}()$ with its position ($\text{Pos}(\text{pub})$) and the cached ER (*cer*) as parameters. When the cached ER is a forged value, function $\text{ISINSL}()$ outputs the overridden value *cer.o_insl* instead of genuine *insl*.

IM protocols abort the protocol execution if malicious behavior is detected. The detection of forged states is modeled by function $\text{ISGEN}(\text{imstate}_x)$ where imstate_x refers to an IM-related state owned by peer x . In CSP#, we model a function with the same name to detect malicious behavior. Aborting protocol execution is modeled by executing $\text{cheat} \rightarrow \text{Stop}$.

We stress that the modeling of malicious behavior should focus on including all possible behaviors even though a behavior is unlikely to benefit the attacker. Whether or not a specific behavior leads to an effective attack should be automatically verified by the model checker rather than determined by human who models the system.

6.2 Property Specifications

In this section, we specify correctness and secrecy-1 properties as reachability tests so they can be automatically verified by the model checker.

6.2.1 Correctness

The correctness property (see Section 3.4.1) is satisfied if and only if the actual protocol output is identical to what an IM is expected to output.

Correctness The output of $\mathcal{P}_s$ (i.e., the actual protocol output a_output) is identical to the result of $f_{im}(gs_{pub}, aoi_{sub})$ (i.e., the expected protocol output e_output).

We have defined that the expected output is denoted by variable e_output and the actual output is denoted by variable a_output . The variable $output_ready$ additionally defines whether the variable a_output is ready to be verified. A correct output can be defined as a state: when the output is ready, the actual output must be identical to the expected output. If an IM scheme satisfies the correctness property, the protocol must always be able to reach the above state. To testify whether the correctness property is always satisfied, we can define its negative proposition *NegCorrectness* and test if the negative state is reachable under benign and HBC behavior (see Figure 6.10). If *NegCorrectness* is reachable (valid), $\mathcal{P}_s$ does not satisfy the correctness property and the event trace leading to the incorrectness will be given; otherwise, if *NegCorrectness* is unreachable (invalid), $\mathcal{P}_s$ satisfies correctness property.

6.2.2 Secrecy-1

The secrecy-1 property (see Section 3.4.2) is satisfied if Alice cannot know Bob's position unless it is authorized.

```

Benign() = benign{beh = BEN; } → LiteGame();
HBC() = hbc{beh = HBC; } → LiteGame();
#define NegCorrectness (output_ready && e_output! = a_output);
#assert Benign reaches NegCorrectness;
#assert HBC reaches NegCorrectness;
    
```

Figure 6.10: Correctness property

```

HBC() = hbc{beh = HBC; } → LiteGame();
Mal() = mal{beh = MAL; } → LiteGame();
#define NegSecrecy1 (k.know(Pos(BOB)) && e_output! = Gs(BOB));
#assert HBC reaches NegSecrecy1;
#assert Mal reaches NegSecrecy1;
    
```

Figure 6.11: Secrecy-1 property

Secrecy-1 Alice knows the Bob’s position (i.e., $k.know(Pos(BOB))$) only if Alice acts as the subscriber and $f_{im}(gs_{bob}, aoi_{alice}) = gs_{bob}$ (i.e., $e_output == Gs(BOB)$).

This property involves two clauses. The first clause is whether or not Alice knows Bob’s position. This can be testified by $k.know(Pos(BOB))$. The second clause is whether or not Alice should know Bob’s position. This can be testified by the condition $e_output == Gs(BOB)$ (Alice is legal to know Bob’s game state and therefore she is allowed to know Bob’s position). If the second clause is satisfied, the first clause must be satisfied. Like what we did in specifying the correctness property, we specify the negative proposition of secrecy-1 as *NegSecrecy1* and test if the negative proposition is reachable by the lite game under HBC and malicious behavior (see Figure 6.11). If *NegSecrecy1* is reachable, $\mathcal{P}_s$ does not satisfy the secrecy-1 property and the event trace leading to the attack will be given; otherwise, if *NegSecrecy1* is unreachable, $\mathcal{P}_s$ satisfies secrecy-1 property.

```

Im(pub, sub) = CS_Srv(pub, sub) ||| CS_Sub(pub, sub);
CS_Srv(pub, sub) = CS_Srv_Benign(pub, sub);
CS_Sub(pub, sub) =
    (CS_Sub_BenignHBC(pub, sub) □ [beh == MAL]CS_Sub_Mal(pub, sub));

CS_Srv_Benign(pub, sub) =
    ifa (ISREL(Gs(pub), Aoi(sub))) {
        srv2sub!Gs(pub) → Skip
    } else {
        nogsmg!0 → Skip
    };

CS_Sub_BenignHBC(pub, sub) =
    (
        srv2sub?gsmg{if (beh == HBC){k.collect(gsmg); }} →
        protocol_output{a_output = gsmg; output_ready = true; } → Skip
    ) interrupt (
        nogsmg?0 →
        protocol_output{a_output = Ud(); output_ready = true; } → Skip
    );

CS_Sub_Mal(pub, sub) =
    (
        srv2sub?gsmg{k.collect(gsmg); } →
        (( protocol_output{a_output = gsmg; output_ready = true; } → Skip ) □
        ( protocol_output{a_output = Ud(); output_ready = true; } → Skip ))
    ) interrupt (
        nogsmg?0 →
        protocol_output{a_output = Ud(); output_ready = true; } → Skip
    );
    
```

 Figure 6.12: Modeling of protocol $\mathcal{P}_{cs}$

6.3 Modeling IM Schemes

In this section, we will individually model IM schemes CS, SP2P, SAFE and PP2P. The full CSP# source code of their models can be found in Appendix B.

6.3.1 CS

The CS protocol $\mathcal{P}_{cs}$ has been described in Section 3.2.1. Figure 6.12 illustrates the CSP# model of protocol $\mathcal{P}_{cs}$. In the C/S infrastructure, the IM protocol is executed between the central server (process $CS_Srv(pub, sub)$) and the subscriber (process $CS_Sub(pub, sub)$). Process $Im(pub, sub)$ models their interaction by interleaving composition. Since the server is maintained by the game operator, its behavior is always benign (process $CS_Srv_Benign(pub, sub)$). The subscriber may have benign, HBC (process $CS_Sub_BenignHBC(pub, sub)$) or malicious behavior (process $CS_Sub_Mal(pub, sub)$). Because of the similarity between the benign and HBC behaviors of the subscriber, we employ one process $CS_Sub_BenignHBC(pub, sub)$ to model both of them. In process $CS_Srv_Benign(pub, sub)$, the server calculates whether the publisher's game state is relevant to the subscriber by invoking $ISREL(Gs(pub), Aoi(sub))$. If it is relevant, the server will send the publisher's game state to the subscriber via channel $srv2sub$. Otherwise, the server will trigger $nogsmg$ signal to tell the subscriber that there is no game state update in this round. This signal is not explicitly defined in the original protocol but is necessary to model this behavior in model checking. In process $CS_Sub_BenignHBC(pub, sub)$, if the game state update message $gsmg$ is received via channel $srv2sub$, benign subscriber will output $gsmg$ ($a_output = gsmg$ and $output_ready = true$). HBC subscriber will additionally collect $gsmg$ as its knowledge k . Otherwise, if $nogsmg$ signal is received, the subscriber will output Ud . The malicious behavior of the subscriber is modeled by another process, $CS_Sub_Mal(pub, sub)$. In this process, the malicious subscriber, even if it receives $gsmg$, may output Ud . The complete CS model can be found in Appendix B.2.

6.3.2 SP2P

The SP2P protocol $\mathcal{P}_{sp2p}$ has been described in Section 3.2.2. Figure 6.13 illustrates the modeling of protocol $\mathcal{P}_{sp2p}$. In the P2P architecture, the IM protocol is executed between

```

Im(pub, sub) = SP2P_Pub(pub, sub) ||| SP2P_Sub(pub, sub);
SP2P_Pub(pub, sub) =
  (SP2P_Pub_BenignHBC(pub, sub) □ [beh == MAL]SP2P_Pub_Mal(pub, sub));
SP2P_Sub(pub, sub) =
  (SP2P_Sub_BenignHBC(pub, sub) □ [beh == MAL]SP2P_Sub_Mal(pub, sub));

SP2P_Pub_BenignHBC(pub, sub) =
  SP2P_Pub_BenignHBC_SlUpdate(pub, sub); SP2P_Pub_BenignHBC_GsUpdate(pub, sub);
SP2P_Pub_Mal(pub, sub) =
  SP2P_Pub_Mal_SlUpdate(pub, sub); SP2P_Pub_Mal_GsUpdate(pub, sub);

SP2P_Sub_BenignHBC(pub, sub) =
  SP2P_Sub_BenignHBC_SlUpdate(pub, sub); SP2P_Sub_BenignHBC_GsUpdate(pub, sub);
SP2P_Sub_Mal(pub, sub) =
  SP2P_Sub_Mal_SlUpdate(pub, sub); SP2P_Sub_Mal_GsUpdate(pub, sub);

...
SP2P_Sub_Mal_SlUpdate(pub, sub) =
  (
    sub2pub!Er(sub) → Skip □
    noermmsg!0 → Skip □
    sub2pub!Er(sub, true) → Skip □
    sub2pub!Er(sub, false) → Skip
  );

SP2P_Sub_Mal_GsUpdate(pub, sub) =
  (
    pub2sub?gsmmsg{k.collect(gsmmsg); } → Skip
  ) interrupt (nogsmmsg?0 → Skip);

SP2P_Pub_Mal_SlUpdate(pub, sub) =
  (sub2pub?ermmsg{k.collect(ermmsg); } → Skip) interrupt (noermmsg?0 → Skip);

SP2P_Pub_Mal_GsUpdate(pub, sub) =
  (
    pub2sub!Gs(pub) → Skip □
    nogsmmsg!0 → Skip □
    pub2sub!Gs(pub, true) → Skip □
    pub2sub!Gs(pub, false) → Skip
  );
    
```

 Figure 6.13: Modeling of protocol $\mathcal{P}_{sp2p}$

the publisher (process $SP2P_Pub(pub, sub)$) and subscriber (process $SP2P_Sub(pub, sub)$). Process $Im(pub, sub)$ models their interaction by interleaving composition. Since $\mathcal{P}_{sp2p}$ is based on P2P architecture, benign, HBC and malicious behavior must be all considered. The publisher/subscriber may have benign, HBC (process $SP2P_Pub_BenignHBC$

$(pub, sub)/SP2P_Sub_BenignHBC(pub, sub)$) or malicious behavior (process $SP2P_Pub_Mal(pub, sub) / SP2P_Sub_Mal(pub, sub)$). Because of the similarity of the benign and HBC behavior, we employ one process to model both of them for each entity (processes $SP2P_Pub_BenignHBC(pub, sub)$ and $SP2P_Sub_BenignHBC(pub, sub)$ for benign/HBC publisher and subscriber respectively). The details of these two processes can be found in Appendix B.3. Process $SP2P_Sub_Mal(pub, sub)$ models a malicious subscriber (Alice). Alice attempts to send genuine ER update message, skip ER update, or send forged ER update message in the SL update (process $SP2P_Sub_Mal_SlUpdate(pub, sub)$). In the state update (process $SP2P_Sub_Mal_GsUpdate(pub, sub)$), she collects the state update message, if any, to her knowledge. Process $SP2P_Pub_Mal(pub, sub)$ models a malicious publisher (Alice). In the SL update (process $SP2P_Pub_Mal_SlUpdate(pub, sub)$), Alice receives ER update message or *noerrmsg* signal similar to what a benign publisher does. But she additionally collects the ER message as part of her knowledge. In the game state update (process $SP2P_Pub_Mal_GsUpdate(pub, sub)$), she attempts to send genuine game state update message ($pub2sub!Gs(pub)$), skip game state update ($nogsmg!0$), or send forged game state update message ($pub2sub!Gs(pub, true) \square pub2sub!Gs(pub, false)$). The complete SP2P model can be found in Appendix B.3.

6.3.3 SAFE

The SAFE protocol $\mathcal{P}_{safe}$ has been described in Chapter 4. Since it is based on P2P infrastructure, all cases (i.e., Alice is benign, HBC, and malicious) must be considered here. The SL update is based on privacy preserving protocol KM-1, which guarantees the confidentiality of the private inputs ER and DR. Moreover, the proposed commitment scheme further guarantees malicious behavior on KM-1 would be detected. In this case, we model SAFE based on the fact that 1) HBC Alice cannot collect sensitive information from KM-1; and 2) malicious Alice cannot perform effective attack on KM-1.

```

Im(pub, sub) = SAFE_Pub(pub, sub) ||| SAFE_Sub(pub, sub);
SAFE_Pub(pub, sub) =
  (SAFE_Pub_BenignHBC(pub, sub) □ [beh == MAL]SAFE_Pub_Mal(pub, sub));
SAFE_Sub(pub, sub) =
  (SAFE_Sub_BenignHBC(pub, sub) □ [beh == MAL]SAFE_Sub_Mal(pub, sub));

SAFE_Pub_BenignHBC(pub, sub) =
  SAFE_Pub_BenignHBC_SlUpdate(pub, sub); SAFE_Pub_BenignHBC_GsUpdate(pub, sub);
SAFE_Pub_Mal(pub, sub) =
  SAFE_Pub_Mal_SlUpdate(pub, sub); SAFE_Pub_Mal_GsUpdate(pub, sub);

SAFE_Sub_BenignHBC(pub, sub) =
  SAFE_Sub_BenignHBC_SlUpdate(pub, sub); SAFE_Sub_BenignHBC_GsUpdate(pub, sub);
SAFE_Sub_Mal(pub, sub) =
  SAFE_Sub_Mal_SlUpdate(pub, sub); SAFE_Sub_Mal_GsUpdate(pub, sub);

...
SAFE_Sub_Mal_SlUpdate(pub, sub) =
  ( sub2pub!ErUpdate → SAFE_Sub_KM1(pub, sub) □
    noerupdate!0 →
    ( SAFE_Sub_KM1(pub, sub)
      ) interrupt (nodrupdate?0 → Skip)
  );

SAFE_Sub_Mal_GsUpdate(pub, sub) = ...

SAFE_Pub_Mal_SlUpdate(pub, sub) =
  ( sub2pub?ErUpdate →
    SAFE_Pub_KM1(pub, sub)
  ) interrupt (noerupdate?0 →
    ( SAFE_Pub_KM1(pub, sub) □
      ndrupdate!0 → Skip
    )
  );

SAFE_Pub_Mal_GsUpdate(pub, sub) = ...
    
```

 Figure 6.14: Modeling of protocol $\mathcal{P}_{safe}$

Figure 6.14 illustrates the behavior of the publisher and subscriber in protocol $\mathcal{P}_{safe}$. The SAFE protocol is executed between the publisher (process $\text{SAFE_Pub}(\text{pub}, \text{sub})$) and subscriber (process $\text{SAFE_Sub}(\text{pub}, \text{sub})$). Again, process $\text{Im}(\text{pub}, \text{sub})$ models their interaction by interleaving composition. The publisher/subscriber may have benign, HBC (process $\text{SAFE_Pub_BenignHBC}(\text{pub}, \text{sub})/\text{SAFE_Sub_BenignHBC}(\text{pub}, \text{sub})$) or malicious behavior (process $\text{SAFE_Pub_Mal}(\text{pub}, \text{sub})/\text{SAFE_Sub_Mal}(\text{pub}, \text{sub})$). Be-

cause of the similarity of the benign and HBC behavior, we employ process $SP2P_Pub_BenignHBC(pub, sub)$ (process $SP2P_Sub_BenignHBC(pub, sub)$) to model benign/HBC behavior of the publisher (subscriber). The details of these two processes can be found in Appendix B.4. In SL update, if Alice is a malicious subscriber (process $SAFE_Sub_Mal_SlUpdate(pub, sub)$), she may execute KM-1 protocol by sending $ErUpdate$ message ($sub2pub!ErUpdate$), or by receiving DR update message (i.e., msg_1 in the SAFE protocol). She may also skip the KM-1 protocol. If Alice is a malicious publisher (process $SAFE_Pub_Mal_SlUpdate(pub, sub)$), she may receive $ErUpdate$ and start to execute KM-1. She may also initiate KM-1 by sending msg_1 (modeled in process $SAFE_Pub_KM1(pub, sub)$). Otherwise, she may skip the KM-1 protocol. For state update, malicious Alice's behavior, as either a subscriber (process $SAFE_Sub_Mal_GsUpdate(pub, sub)$) or a publisher (process $SAFE_Pub_Mal_GsUpdate(pub, sub)$), is similar to that in SP2P. The source code of SAFE model can be found in Appendix B.4.

6.3.4 PP2P

The PP2P protocol $\mathcal{P}_{pp2p}$ has been described in Chapter 5. PP2P is also based on P2P architecture and therefore, benign, HBC and malicious behavior (of Alice) must be all considered. The core of PP2P is to delegate the processing of sensitive information to a trusted block on each peer. The verification of PP2P depends on the following properties of trusted block, which have been proven in a separate piece of work [60].

- (i) The trusted block is isolated from the legacy system.
- (ii) The trusted block can interact with the legacy system.
- (iii) The trusted block is able to securely communicate with another trusted block.

Figure 6.15 illustrates the modeling of protocol $\mathcal{P}_{pp2p}$. The PP2P protocol is executed among the publisher's legacy and trusted IMs (processes $PP2P_Pub_Legacy(pub, sub)$

$$\begin{aligned}
Im(pub, sub) &= PP2P_Pub_Legacy(pub, sub) ||| PP2P_Sub_Legacy(pub, sub) ||| \\
&\quad PP2P_Pub_TB(pub, sub) ||| PP2P_Sub_TB(pub, sub); \\
PP2P_Pub_Legacy(pub, sub) &= \\
&\quad (PP2P_Pub_Legacy_BenignHBC(pub, sub) \square [beh == MAL] PP2P_Pub_Legacy_Mal(pub, sub)); \\
PP2P_Sub_Legacy(pub, sub) &= \\
&\quad (PP2P_Sub_Legacy_BenignHBC(pub, sub) \square [beh == MAL] PP2P_Sub_Legacy_Mal(pub, sub)); \\
PP2P_Pub_Legacy_BenignHBC(pub, sub) &= \\
&\quad PP2P_Pub_Legacy_BenignHBC_SIUpdate(pub, sub); \\
&\quad PP2P_Pub_Legacy_BenignHBC_GsUpdate(pub, sub); \\
PP2P_Pub_Legacy_Mal(pub, sub) &= \\
&\quad PP2P_Pub_Legacy_Mal_SIUpdate(pub, sub); \\
&\quad PP2P_Pub_Legacy_Mal_GsUpdate(pub, sub); \\
PP2P_Sub_Legacy_BenignHBC(pub, sub) &= \\
&\quad PP2P_Sub_Legacy_BenignHBC_SIUpdate(pub, sub); \\
&\quad PP2P_Sub_Legacy_BenignHBC_GsUpdate(pub, sub); \\
PP2P_Sub_Legacy_Mal(pub, sub) &= \\
&\quad PP2P_Sub_Legacy_Mal_SIUpdate(pub, sub); \\
&\quad PP2P_Sub_Legacy_Mal_GsUpdate(pub, sub); \\
&\dots
\end{aligned}$$
Figure 6.15: Modeling of protocol $\mathcal{P}_{pp2p}$

and $PP2P_Pub_TB(pub, sub)$ respectively), and the subscriber's legacy and trusted IMs (processes $PP2P_Sub_Legacy(pub, sub)$ and $PP2P_Sub_TB(pub, sub)$ respectively). Process $Im(pub, sub)$ models their interactions by interleaving composition. The publisher's and subscriber's legacy IMs may have benign, HBC (processes $PP2P_Pub_Legacy_BenignHBC(pub, sub)$ and $PP2P_Sub_Legacy_BenignHBC(pub, sub)$) or malicious behavior (processes $PP2P_Pub_Legacy_Mal(pub, sub)$ and $PP2P_Sub_Legacy_Mal(pub, sub)$). Because of the similarity of the benign and HBC behavior, we employ process $SP2P_Pub_BenignHBC(pub, sub)$ (process $SP2P_Sub_BenignHBC(pub, sub)$) to model both benign and HBC behavior of the publisher (subscriber).

We model the trusted IM on each peer as an independent CSP# process (processes $PP2P_Pub_TB(pub, sub)$ and $PP2P_Sub_TB(pub, sub)$) running in parallel with the legacy IM. The behavior of trusted blocks is always benign. The public functions defined in the trusted block are modeled as CSP processes which can be communicated via syn-

```

channel tb_genermsg 0;
channel tb_revealstate 0;
PP2P_Sub_TB(pub, sub) =
(
  (
    tb_genermsg?pub.er →
    tb_genermsg!(PACK(er, tb_signk[sub], tb_enck[sub])) → Skip
  ) □
  (
    tb_revealgs?pub.gsmg.aoi {
      game_state = UNPACK(gsmg, tb_verifyk[sub], tb_deck[sub]);
    } →
    ifa (game_state == Error || !ISGEN(game_state) || ISGEN(aoi)) {
      cheat → Stop
    } else ifa (ISREL(game_state, aoi)) {
      tb_revealgs!game_state → Skip
    } else {
      tb_revealgs!Ud() → Skip
    }
  )
);
PP2P_Sub_TB(pub, sub);

```

Figure 6.16: Modeling of subscriber's trusted block (*sub_trust*) in protocol $\mathcal{P}_{pp2p}$

chronous channels. For packing operation, we define function $\text{PACK}(\text{payload}, \text{signk}, \text{enck})$ to generate message by using *payload* with *signk* as the signing key and *enck* as the encryption key. For unpacking operation, we define function $\text{UNPACK}(\text{pmsg}, \text{verifyk}, \text{deck})$ to unpack payload from message *pmsg* with *verifyk* as the verification key and *enck* as the decryption key. In case the payload cannot be unpacked because of wrong keys or broken message, the function returns *Error*.

Figure 6.16 illustrates an example of how the subscriber's trusted block is modeled. Two synchronous channels *tb_genermsg* and *tb_revealstate* model the entry points of public functions $\text{GENERMSG}(\text{pub}, \text{er})$ and $\text{REVEALSTATE}(\text{pub}, \text{gsmg}, \text{aoi})$ respectively (see Section 5.2.1 for details of these public functions). The subscriber's legacy IM can

Table 6.3: Verification results of correctness and secrecy-1

Scheme	Property	Behavior	Result	#State	#Transition	Time(s)
CS	<i>NegCorrectness</i>	benign	invalid	62	65	< 1
		HBC	invalid	62	65	< 1
	<i>NegSecrecy1</i>	HBC	invalid	62	65	< 1
		malicious	invalid	73	83	< 1
SP2P	<i>NegCorrectness</i>	benign	invalid	1619	1834	< 1
		HBC	invalid	2013	2242	< 1
	<i>NegSecrecy1</i>	HBC	valid	58	61	< 1
		malicious	valid	64	67	< 1
SAFE	<i>NegCorrectness</i>	benign	invalid	11783	14624	< 1
		HBC	invalid	11783	14624	< 1
	<i>NegSecrecy1</i>	HBC	valid	155	170	< 1
		malicious	valid	463	550	< 1
PP2P	<i>NegCorrectness</i>	benign	invalid	4480	4956	< 1
		HBC	invalid	5408	5900	< 1
	<i>NegSecrecy1</i>	HBC	invalid	5408	5900	< 1
		malicious	invalid	9779	10558	< 1

thereby invoke a public function by sending parameters to the corresponding channel and waiting for the result from the same channel.

The adversary behavior only applies to the legacy IMs. In addition to the messages received, the legacy IM with HBC behavior also collects results from public functions. Malicious Alice may also invoke public functions with various parameters, attempting to retrieve more information. The source code of PP2P model can be found in Appendix B.5.

6.4 Experiment Results and Discussions

In this section, we present the experiment results of formal verification, which are summarized in Table 6.3. Particularly, evaluation covers correctness and secrecy-1 properties. For those unsatisfied properties, counter examples are given and analyzed.

... $\rightarrow$ *pubpos_notin_subaoi* $\rightarrow$ *subaoi_notin_suber* $\rightarrow$ *aliceispub* $\rightarrow$
 ... $\rightarrow$ *updateer* $\rightarrow$... $\rightarrow$ *sub2pub*.*Er*(*BOB*)

Figure 6.17: The counter example leading $\mathcal{P}_{sp2p}$ to *NegSecrecy1*

6.4.1 Correctness

According to the experiment results, property *NegCorrectness* is invalid for all IM schemes. This in turn concludes that all IM schemes satisfy correctness property with regard to both benign and HBC Alice. Note that this property does not apply to malicious Alice because malicious behavior is expected to violate correctness. The results indicate all schemes have correct functionality of what f_{im} defines, and HBC behavior does not change the correctness of protocols. Because of the abstraction of IM-related states, the numbers of states and transitions are reduced without losing fidelity. In terms of execution time, all verifications can be finished within one second.

6.4.2 Secrecy-1

As expected, the CS protocol has invalid result of *NegSecrecy1* property. Among three P2P IM schemes, only PP2P has invalid result of *NegSecrecy1* property. This means only CS and PP2P protocols satisfy secrecy-1. We thus inspect counter examples generated by the model checker for SP2P and SAFE schemes.

For protocol $\mathcal{P}_{sp2p}$, *NegSecrecy1* is valid for both HBC and malicious Alice, which indicates the SP2P does not satisfy secrecy-1. In particular, the model checker generates a minimal counter example that reaches *NegSecrecy1* for both HBC and malicious Alice. Part of the event trace in this counter example is shown in Figure 6.17. The example shows that Alice acts as the publisher and Bob acts as the subscriber (event *aliceispub*). The IM protocol starts in the situation when Alice's position is outside Bob's AOI (event *pubpos_notin_subaoi*) and Bob's AOI is not covered by Bob's ER

$$\begin{aligned} & \dots \rightarrow \text{pubpos_notin_subaoi} \rightarrow \text{subaoi_notin_suber} \rightarrow \text{bobispub} \rightarrow \\ & \dots \rightarrow \text{sub_in_pubsl} \rightarrow \dots \rightarrow \text{pub2sub.Gs(BOB)} \end{aligned}$$
Figure 6.18: The counter example leading $\mathcal{P}_{safe}$ to $NegSecrecy1$

(event *subaoi_notin_suber*). In this case, the expected output should be *Ud()*. However, when Bob updates his ER (event *updateer*) and sends ER update message to Alice (event *sub2pub.Er(BOB)*), Alice knows Bob's ER and in turn his position. This violates secrecy-1. This result is as expected because SP2P is known to expose ER to the publisher.

For protocol $\mathcal{P}_{safe}$, *NegSecrecy1* is valid for both HBC and malicious Alice, which indicates the SAFE does not satisfy secrecy-1 either. In particular, the model checker generates a minimal counter example that violates secrecy-1. Partial trace of events in this counter example is shown in Figure 6.18. The example shows that Alice acts as the subscriber and Bob acts as the publisher (event *bobispub*). The IM protocol starts in the case when Bob's position is outside Alice's AOI (event *pubpos_notin_subaoi*) and Alice's AOI is not covered by her ER (event *subaoi_notin_suber*). Under this circumstance, the expected output should be *Ud()*. However, since Alice is in Bob's SL (event *sub_in_pubsl*), Bob sends his game state to Alice (event *pub2sub.Gs(BOB)*). HBC or malicious Alice therefore knows Bob's game state and his position. This violates secrecy-1. This result is as expected because SAFE does not protect the game state update.

6.5 Summary

In this chapter, we have presented formal verification of correctness and secrecy-1 properties for IM schemes. The verification is performed by model checking. Particularly, the behaviors of the lite game and adversary are modeled. Properties being verified are specified as specifications in CSP#. The system model and property specifications are

both inputs to the model checker for verification. The experiment results show that all IM schemes achieve correctness property. However, among P2P based IM schemes, only PP2P satisfies secrecy-1 property. We will continue to evaluate other properties such as secrecy-2 and bandwidth usage in the next chapter.

Chapter 7

Quantitative Evaluations

In this chapter, we apply simulation-based approach to evaluate secrecy-2 and bandwidth usage which are two quantitative properties of an IM schemes. We start by describing how the simulator models each peer's states and changes in the game. Subsequently, we detail how secrecy-2 and bandwidth usage are calculated based on simulation statistics. Specially, we model the attacker's behavior on predicting the victim's position for evaluating secrecy-2. After that, we present experiment results of four IM schemes, i.e., CS, SP2P, SAFE, and PP2P. The comparisons between the four IM schemes are given. The content of this chapter is derived from our published paper 'Measuring Information Exposure Attacks on Interest Management' [59].

Both model checking and simulation approaches involve exploring system states. However, they deal with nondeterministic choices in different ways. Model checking is designed to explore all possible choices, but certain abstraction (e.g., symbolic representation of states) must be applied to avoid state explosion. Therefore, it is suitable to verify functional properties such as correctness and secrecy-1. On the contrary, the simulation approach usually selects one choice in a stochastic manner and it relies on repeated trials to obtain reliable statistics. Since secrecy-2 and bandwidth usage are quantitative properties, the evaluation requires numerical representations of states and exploring all numerical values is computationally infeasible for model checking.

Table 7.1: Definitions of IM-related states and functions in simulation

Notation/fuction	Definition
V	$\{(x, y) : x, y \in \{1, 2, \dots, len_v\}\}$
pos_x	$p : p \in V$
aoi_x	$cr(pos_x, ra_{aoi})$
er_x	$cr(p, ra_{er})$
dr_x	$cr(p, ra_{dr})$
gs_x	pos_x
$GETAOI(pos_x)$	Returns $cr(pos_x, ra_{aoi})$
$GETER(p)$	Returns $cr(p, ra_{er})$
$GETDR(p)$	Returns $cr(p, ra_{dr})$
$GETGs(p)$	Returns pos_x
$GETPOS(cr(cp, ra))$	Returns cp

7.1 Game Simulation

The game simulation is to study each peer's IM-related states and behaviors in the lite game when a specific IM scheme is applied. The sizes and the shapes of VE, AOI, ER and DR will be explicitly defined. The relationships and functions among them will be also detailed. Besides, we will model how the peers move in the game.

7.1.1 Simulating IM-related States

Recall the lite game defined in Section 3.1.4, peers Alice and Bob play the game round by round. Their behaviors are modeled in Chapter 6 in a symbolic manner, i.e., VE and IM-related states are modeled as symbols. When it comes to simulation, we must model these states explicitly.

Without losing generality, the VE (denoted by V) is modeled as a square region in 2D space. This region is defined as a finite set of discrete coordinates. For each dimension of coordinate, the value can be integers from 1 to len_v . A peer x 's position pos_x is an element in this set. A peer x 's AOI aoi_x is defined as a circular region centered at the peer's position pos_x with radius of ra_{aoi} . Similarly, the ER and DR are also defined as circular regions centered at the position p_c with radiuses of ra_{er} and ra_{dr} respectively.

Since circular region is commonly used, we define it as

$$cr(cp, ra) : \{p : \| p, cp \| \leq ra \wedge p \in V\}$$

where cp is the center position, ra is the radius and the operator $\| p_1, p_2 \|$ computes the Euclidean distance between coordinates p_1 and p_2 . Therefore, a peer x 's AOI, ER and DR are defined as $cr(pos_x, ra_{aoi})$, $cr(p, ra_{er})$, and $cr(p, ra_{dr})$ respectively. They can be calculated by invoking functions $GETAOI(pos_x)$, $GETER(p)$, and $GETDR(p)$ respectively. A peer x 's game state gs_x contains its position pos_x as well as other game-specific information unrelated to IM. Function $GETPOS(cr(cp, ra))$ returns the center position cp used to construct the circular region. Table 7.1 summarizes the above definitions. The relationships among these states such as $p \in R$, $R_1 \subset R_2$, $R_1 \cap R_2$ and $R_1 - R_2$ can be calculated by their original mathematical definitions (p represents a position and R represents a region).

7.1.2 Mobility Model

The mobility model defines how the peers move in the simulation, which is the first step in the lite game in each round. A peer is allowed to move to one of eight adjacent positions in every round. In our work, the direction of movement follows the random waypoint model [22], which is defined as:

- (i) The peer randomly chooses a target position in the VE as the destination.
- (ii) The peer walks towards the destination round by round.
- (iii) When the peer reaches the destination, goto step (i).

Specifically, every peer moves independently and peers may share the same position with each other.

7.2 Property Evaluation

In this section, we describe how properties of secrecy-2 and bandwidth usage are evaluated via simulation. Particularly, the evaluation of secrecy-2 requires to simulate attacker's estimation of victim's position. This is achieved by Monte Carlo based simulation [113].

7.2.1 Secrecy-2

Secrecy-2 property quantitatively characterizes Alice's knowledge on Bob's position.

Secrecy-2 Alice predicts Bob's position as a random variable X and secrecy-2 is defined as the average Shannon entropy of X per round.

In other words, this property quantitatively captures Alice's knowledge on where Bob is. Alice's knowledge on Bob's position is modeled as a random variable X with probability mass function (PMF)

$$f_X(p) = Pr(X = p)$$

which predicts the probability of 'Bob is at position p ' for all possible p over the entire VE. The Shannon entropy of X in round i is thus calculated as

$$entropy_i = - \sum_{p \in V} f_X(p) \log_2 f_X(p) \quad (\text{bits})$$

and the secrecy-2 is defined as $\overline{entropy}$ (i.e., the average entropy per round).

The key to calculate secrecy-2 is to estimate the PMF. To this end, we employ a Monte Carlo based approach. We define a *pseudo peer* as a possible estimation of Bob's states such as where he is and whether he updates his ER. Estimations of Bob's position (i.e., pseudo peers) are randomly generated in the VE. Attacker's (i.e., Alice's) knowledge is used to further refine the estimation. In the simulation, a large number of pseudo peers will be simulated. Pseudo peers move in the VE with respect to the mobility model like

Figure 7.1: The visualization of pseudo peers (SAFE)

Bob does. Since the waypoints are randomly generated, pseudo peers with the same initial states may eventually move to different places in the VE. In every round, Alice (the attacker) obtains information that where Bob could be or could not be, and whether he just updates ER or DR. Pseudo peers whose states conflict with obtained information will be removed. This is referred to as *refinement operation*. Function $f_X(p)$ is estimated as the percentage of the number of pseudo peers fallen on position p over the total number of pseudo peers.

A visualization of pseudo peers under SAFE protocol is shown in Figure 7.1. The green and blue dots represent the actual positions of Alice and Bob respectively. The yellow ring is the AOI and two gray rings are ER (smaller) and DR (larger) respectively. The red dots are the pseudo peers' positions. The darker an area, the more pseudo peers are in the area.

Table 7.2: Refinement operations

Catalogue	Knowledge	Remove pp if unsatisfied
Position	ATPOS(p)	$pos_{pp} = p$
	INREGION(R)	$pos_{pp} \in R$
	NOTINREGION(R)	$pos_{pp} \notin R$
Region	ERCONTAINS(p)	$p \in er_{pp}$
	ERNOTCONTAIN(p)	$p \notin er_{pp}$
	EROVERLAPS(R)	$er_{pp} \wedge R \neq \emptyset$
	ERNOTOVERLAP(R)	$er_{pp} \wedge R = \emptyset$
	DROVERLAPS(R)	$dr_{pp} \wedge R \neq \emptyset$
	DRNOTOVERLAP(R)	$dr_{pp} \wedge R = \emptyset$
Action	JUSTERUPDATE()	er_{pp} is updated in the current round
	NOERUPDATE()	er_{pp} is unchanged in the current round
	JUSTDRUPDATE()	dr_{pp} is updated in the current round
	NODRUPDATE()	dr_{pp} is unchanged in the current round

The refinement is to remove pseudo peers whose states conflict with the attacker's knowledge. Based on different knowledge collected by the attacker, we summarize all possible refinement operations in Table 7.2. Knowledge is modeled as functions. They are grouped into the following catalogues: position-based, region-based and action-based knowledge. The position-based knowledge involves information regarding peer's position. For instance, ATPOS(p) indicates that the attacker knows the victim is at an exact position p . When this knowledge is applied, pseudo peers whose positions are not at position p will be removed. The region-based knowledge involves information regarding peer's region such as ER and DR. For example, ERCONTAINS(p) indicates that the attacker knows the victim's ER contains position p and hence pseudo peers whose ER does not contain position p will be removed. The action-based knowledge involves information regarding peer's actions in the current round. For example, JUSTERUPDATE() indicates that the attacker knows the victim updates its ER in the current round, so pseudo peers who do not update their ERs in the current round will be removed.

Table 7.3: Typical lengths of basic elements

Element	Remark
l_{er}	The length of ER update payload
l_{gs}	The length of game state update payload
l_{crypt}	The length of encryption/signing function output
l_{hash}	The length of hash function output

Table 7.4: Information learned in protocol $\mathcal{P}_{cs}$

Alice's role	Condition	Refinement operation
Publisher	N/A	N/A
Subscriber	msg_{gs} is received	$ATPOS(pos_{bob})$
	msg_{gs} is not received	$NOTINREGION(aoi_{alice})$

7.2.2 Bandwidth Usage

By definition, the bandwidth usage counts all traffics, incurred by an IM protocol, over the Internet. To calculate the bandwidth usage of IM schemes in our work, we record the number of different types of messages transmitted over the Internet. Since the length of each type of message is known, we can then calculate the average bandwidth usage per round. Each type of message may consist of several basic elements. Definitions of these basic elements are given in Table 7.3.

7.3 Modeling IM Schemes

In this section, we will analyze four IM schemes for 1) information obtained in the IE attacks and 2) the length of each transmitted message. They are vital information for quantitatively measuring secrecy-2 and bandwidth usage respectively.

7.3.1 CS

The CS protocol $\mathcal{P}_{cs}$ has been described in Section 3.2.1.

Table 7.5: Information learned in protocol $\mathcal{P}_{sp2p}$

Alice's role	Condition	Refinement operation
Publisher	msg_{er} is received	JUSTERUPDATE(), ATPOS(pos_{bob})
	msg_{er} is not received	NOERUPDATE(), INREGION(er_{bob})
	$insl = true$	ERCONTAINS(pos_{alice})
	$insl = false$	ERNOTCONTAIN(pos_{alice})
Subscriber	msg_{gs} is received	ATPOS(pos_{bob})
	msg_{gs} is not received	NOTINREGION(er_{alice})

7.3.1.1 Secrecy-2

For secrecy-2, Table 7.4 summarizes what information can be collected by Alice in different conditions. When HBC Alice is the publisher, she will not participate in the protocol and thereby will know nothing about Bob's position. When HBC Alice is the subscriber, she will know Bob's position if the game state update is received (step (ii)(a) in Figure 3.2); otherwise, she will know Bob's position is not within her AOI. The malicious Alice receives the same messages compared with what HBC Alice does. So, malicious Alice cannot obtain more information than HBC Alice does.

7.3.1.2 Bandwidth Usage

The protocol only transmits protocol output gs'_{pub} or Ud via Internet. The output gs'_{pub} includes only game state update whose length is l_{gs} . The output Ud has negligible length.

7.3.2 SP2P

The SP2P protocol $\mathcal{P}_{sp2p}$ has been described in Section 3.2.2.

7.3.2.1 Secrecy-2

For secrecy-2, we analyze Alice's behavior. Firstly, we consider that Alice acts as an HBC publisher. If Alice receives msg_{er} (step (ii)(a) in Figure 3.3), she will know received ER is newly updated by Bob (subscriber) and therefore she can obtain ER's center point

as Bob's position. If she does not receive msg_{er} , Alice will know Bob's ER is unchanged and Bob's position is inside his ER. If $insl = true$ (step (ii)(c) in Figure 3.3), Alice will know Bob's ER covers her position; otherwise if $insl = false$, Alice will know Bob's ER does not cover her position. When Alice acts as an HBC subscriber, she can obtain Bob's information in the stage of game state update. If she receives the game state update msg_{gs} from the publisher (Bob), Alice can extract Bob's position from msg_{gs} (step (iii)(a) in Figure 3.3). If she does not receive the game state update msg_{gs} , Alice will know Bob's position is not covered by her ER.

When Alice acts as a malicious publisher, she can learn everything that an HBC publisher can obtain. In addition, Alice may forge message msg_{gs} and send it to Bob (step (ii)(c) in Figure 3.3). However, even if Alice does so, she cannot obtain more information from this attack. Moreover, Bob can detect this forged message and abort the protocol, making the attack not effective. When Alice acts as a malicious subscriber, again she can learn everything that an HBC subscriber can obtain. Moreover, Alice may forge message msg_{er} and send it to Bob (step (i)(b)(2) in Figure 3.3). However, this forged message can be detected by Bob, making the attack not effective either. So, malicious Alice cannot obtain more information than HBC Alice does. Therefore, we will simulate Alice with HBC behavior only. Table 7.4 summarizes what information can be collected by HBC Alice under different conditions, and the corresponding refinement operations that can be applied.

7.3.2.2 Bandwidth Usage

The protocol transmits ER update message msg_{er} and game state update message msg_{gs} via Internet. Like CS scheme, message msg_{gs} only includes game state update, whose length is l_{gs} . Message msg_{er} carries only ER update payload, whose length is l_{er} .

Table 7.6: Information learned in protocol $\mathcal{P}_{safe}$

Alice's role	Condition	Refinement operation
Publisher	msg_{er} is received msg_{er} is not received $insl = true$ $insl = false$	$JUSTERUPDATE()$ $NOERUPDATE()$ $EROVERLAPS(dr_{alice})$ $ERNOTOVERLAP(dr_{alice})$
Subscriber	msg_1 is received and no msg_{er} is sent msg_1 is not received msg_{gs} is received msg_{gs} is not received	$JUSTDRUPDATE()$ $NODRUPDATE()$ $ATPOS(pos_{bob})$ $DRNOTOVERLAP(er_{alice})$

7.3.3 SAFE

The SAFE protocol $\mathcal{P}_{safe}$ has been described in Section 4.3.

7.3.3.1 Secrecy-2

For secrecy-2, we firstly analyze Alice's HBC behavior. When Alice acts as a publisher, she knows whether or not Bob updates his ER (step (ii) in Figure 4.4). Alice also learns $insl$ from the SL update (step (iv)(b) in Figure 4.4). When Alice acts as a subscriber, she knows whether or not the SL update is triggered by Bob due to DR update (step (iii) in Figure 4.4). If the SL update is not triggered by Bob (i.e., msg_1 not received), she will know Bob's DR is unchanged in the current round. Similar to SP2P, Alice will know Bob's position if she receives game state update msg_{gs} (step (v) in Figure 4.4), even if the update is irrelevant. If msg_{gs} is not received, Alice will know Bob's DR does not overlap with her ER.

When Alice is malicious, she can learn everything that an HBC Alice can obtain. If she tries to compromise the execution of SL update, Bob can detect it with the verification protocol (see Figures 4.2 and 4.3), making the attack not effective. The game state update of SAFE (step (iv)(c) to step (v)(b) in Figure 4.4) is the same as that of SP2P, and therefore no effective malicious attack can be performed either. So, again malicious Alice cannot obtain more information than HBC Alice does. Therefore, we will simulate

Table 7.7: The lengths of major messages in protocol $\mathcal{P}_{safe}$

Message	Length
msg_{er}	negligible
msg_1, com_1	$ \Omega l_{crypt} + l_{hash}$
msg_2, com_2	$l_{crypt} + l_{hash}$
msg_{gs}	l_{gs}

 Table 7.8: Information learned in protocol $\mathcal{P}_{pp2p}$

Alice's role	Condition	Refinement operation
Publisher	msg_{er} is received	JUSTERUPDATE()
	msg_{er} is not received	NOERUPDATE()
	$insl = true$	ERCONTAINS(pos_{alice})
	$insl = false$	ERNOTCONTAIN(pos_{alice})
Subscriber	msg_{gs} is received	INREGION(er_{alice})
	msg_{gs} is not received	NOTINREGION(er_{alice})
	Outputs gs_{pub}	ATPOS(pos_{bob})
	Outputs Ud	NOTINREGION(aoi_{alice})

Alice with only the HBC behavior in the protocol $\mathcal{P}_{safe}$. Table 7.6 summarizes what information can be collected by an HBC Alice in different conditions, and the corresponding refinement operations that can be applied.

7.3.3.2 Bandwidth Usage

The lengths of major messages are listed in Table 7.7. Message msg_{er} is only a signal and its length is negligible. The length of msg_1 is $|\Omega| l_{crypt}$ and the length of com_1 is l_{hash} . The length of msg_2 is l_{crypt} and the length of com_2 is l_{hash} .

7.3.4 PP2P

The PP2P protocol $\mathcal{P}_{pp2p}$ has been described in Section 5.2.

7.3.4.1 Secrecy-2

For secrecy-2, we firstly analyze Alice's HBC behavior. When Alice acts as a publisher, she will know Bob updates his ER if she receives ER update message msg_{er} (step (ii)(a)

in Figure 5.8). Otherwise, she will know Bob's ER is unchanged if she does not receive msg_{er} . From the value of $insl$ (step (ii)(b) in Figure 5.8), Alice will know whether Bob's ER covers her position. When Alice acts as a subscriber, she will know Bob's position is covered by her ER if she receives game state update message msg_{gs} (step (iii)(a) in Figure 5.8); otherwise, she will know Bob's position is outside her ER. If the protocol outputs Bob's game state, Alice will know Bob's position; otherwise, she will know Bob's position is outside her AOI.

When Alice is malicious, she can learn everything that an HBC Alice can obtain. If she is a malicious publisher, she may invoke public function $INSL(sub, pos_{pub})$ with forged input pos_{pub} to probe Bob's ER (step (ii)(b) in Figure 5.8). However, the public function $INSL()$ is designed to verify the input parameter and is able to detect this attack, making it ineffective. If Alice is a malicious subscriber, she can forge her ER er_{sub} when invoking $msg_{er} = GENERMSG(pub, er_{sub})$ (step (i)(b)(2) in Figure 5.8). However, this forged ER will be detected by Bob when he invokes public function $CACHEER(sub, msg_{er})$ (step (ii)(a)(1) in Figure 5.8). This makes the attack ineffective. If malicious Alice forges an ER message without invoking public function $msg_{er} = GENERMSG(pub, er_{sub})$, the forged message will not be signed by the trusted block. This abnormal message will be detected by function $CACHEER(sub, msg_{er})$ when it unpacks the message (step (ii)(a)(1) in Figure 5.8). So, this attack is also ineffective. At step (iii) in Figure 5.8, malicious Alice may invoke public function $REVEALGS(pub, msg_{gs}, aoi_{sub})$, with forged aoi_{sub} , attempting to pretend that Bob's position is inside the Alice's AOI. However, there is a mechanism in the public function $REVEALGS()$ to verify the input parameter and thus this attack can be detected. In summary, malicious Alice cannot obtain more information than HBC Alice does. Consequently, we will simulate Alice with HBC behavior only in protocol $\mathcal{P}_{pp2p}$. Table 7.8 summarizes what information can be collected by HBC Alice under different conditions, and the corresponding refinement operations that can be applied.

Table 7.9: The lengths of major messages in protocol $\mathcal{P}_{pp2p}$

Message	Length
msg_{er}	$l_{er} + l_{hash}$
msg_{gs}	$l_{gs} + l_{hash}$

Table 7.10: Typical values for the simulation parameters

Parameters	Value	Remark
len_v	500 units	The length of VE
ra_{aoi}	10 units	The radius of AOI
ra_{er}	25 units	The radius of ER
ra_{dr}	15 units	The radius of DR
l_{er}	4 bytes	The length of ER update payload
l_{gs}	2 KB	The length of game state update payload
l_{crypt}	256 bits	The length of encryption/decryption output
l_{hash}	160 bits	The length of hash function output
$ \Omega $	2,500	The size of Ω (used in SAFE)

7.3.4.2 Bandwidth Usage

PP2P transmits ER update and game state update messages via Internet. The ER update message msg_{er} essentially contains encrypted subscriber's ER with the trusted block's signature, whose length is $l_{er} + l_{hash}$. Similarly, the game state update message msg_{gs} contains encrypted publisher's game state update signed by the trusted block, whose length is $l_{gs} + l_{hash}$. The lengths of major messages in protocol $\mathcal{P}_{pp2p}$ are summarized in Table 7.9.

7.4 Experiment Results and Discussions

In this section, we will present the experiment results from the simulation. The properties of secrecy-2 and bandwidth usage of four IM schemes will be shown.

7.4.1 Simulation Parameters

Before running the simulation, the parameters involved in the simulation must be set. The typical values to these parameters are assigned as follows. The length of VE is 500

Figure 7.2: Secrecy-2 of IM Schemes (the higher the better)

units. The radius of AOI, ER and DR are 10 units, 20 units and 15 units respectively. For numerically calculating the bandwidth, the lengths of basic elements are set. ER update and game state update messages are assumed to take 4 bytes and 2 KB respectively. The encryption/signing uses a 256-bit algorithm and the hashing outputs digits with 160 bits. For the SAFE scheme, we group every (10×10) adjacent points into a cell and $|\Omega|$ is thereby set to 2,500.

7.4.2 Secrecy-2

Figure 7.2 illustrates the simulation results for secrecy-2 property (entropy). The CS scheme is considered as the baseline test because no IE attacks can be performed on this scheme. The average entropy value of CS scheme, 17.12 bits, is regarded as the upper bound. That is, if there is no IE attack, the uncertainty of the Bob's position can be encoded by 17.12 bits on average. Our proposed SAFE and PP2P IM protocols have entropies close to what CS has. The entropy differences of SAFE and PP2P compared to that of CS are 1.91 bits and 1.46 bits respectively. These results are much better than the entropy difference between SP2P and CS, which is 12.59 bits.

The results can be explained by analyzing the protocols. The SP2P scheme has worst mitigation to IE attacks because it transfers both ER update message and game state update message in plaintext. It allows the attacker to reveal the victim's position

Table 7.11: Bandwidth usage of IM schemes

Scheme	Message	#msg sent/round	Bandwidth/msg	Bandwidth/round(bits)
CS	gs_{pub}	0.0013	l_{gs}	21
	Total			21
SP2P	msg_{er}	0.0710	l_{er}	2
	msg_{gs}	0.0086	l_{gs}	141
	Total			143
SAFE	msg_1, com_1	0.0710	$ \Omega l_{crypt} + l_{hash}$	45,451
	msg_2, com_2	0.0710	$l_{crypt} + l_{hash}$	30
	msg_{gs}	0.0260	l_{gs}	426
	Total			45,907
PP2P	msg_{er}	0.0710	$l_{er} + l_{crypt}$	20
	msg_{gs}	0.0086	$l_{gs} + l_{crypt}$	143
	Total			163

easily. The SAFE scheme employs protection in the SL update stage, which guarantees subscriber's ER or publisher's DR cannot be revealed by the attacker. It makes the SAFE scheme much better than the SP2P. The PP2P scheme relies on trusted block to protect private data in both SL and game state update stages. This makes PP2P better than SAFE with regard to secrecy-2.

7.4.3 Bandwidth Usage

The bandwidth usage of each protocol is illustrated in Table 7.11. The data in column ‘#msg sent/round’ are measured by experiment results. The data in Column ‘Bandwidth/msg’ summarize the lengths of the messages. The data in column ‘Bandwidth/round (bits)’ are calculated by the products of above data. On average, the CS scheme has lowest bandwidth usage among all schemes, which is 21 bits per round. The SP2P scheme has lowest bandwidth usage among all P2P-based schemes, which is 143 bits per round. The PP2P scheme requires little more bandwidth usage because of the inclusion of signature in messages. It takes 163 bits per round. The SAFE scheme, however, requires 45,907 bits per round. This is significantly larger than other protocols. The high communication cost is mainly due to the privacy preserving protocol.

Table 7.12: Properties of IM Schemes

Scheme	Correctness	Secrecy-1	Secrecy-2	Bandwidth	HW Req
CS	✓	N/A	17.12	21	
SP2P	✓	×	4.53	143	
SAFE	✓	×	15.21	45,907	
PP2P	✓	✓	15.66	163	Virtualization, TC

7.4.4 Discussions

We have evaluated the properties for four IM schemes including correctness, secrecy-1, secrecy-2 and bandwidth usage. Table 7.12 summarizes these properties of the four IM schemes. The SP2P scheme is very vulnerable to IE attacks. Its lowest entropy (secrecy-2) shows that the victim player’s position is highly predictable to the attacker. Any game that employs SP2P as the IM scheme may face severe cheating issues.

For the selection between SAFE and PP2P, various aspects must be taken into considerations. Generally, PP2P outperforms SAFE regarding both the mitigation to IE attacks (secrecy-1 and secrecy-2) and network performance (bandwidth usage). However, PP2P requires specific hardware such as virtualization and trusted computing to establish the protection provided by the trusted block. Currently, not every desktop computer satisfies this requirement. For SAFE, the major problem is the communication cost. The execution of KM-1 protocol requires three magnitudes higher bandwidth compared to what PP2P does. To optimize bandwidth usage of SAFE, we may decrease the size of Ω to reduce bandwidth usage per message, or increase the size of ER and DR to reduce the number of ER/DR updates.

7.5 Summary

In this chapter, we have presented quantitative evaluation of secrecy-2 and bandwidth usage for IM schemes. To perform this evaluation, we employ a simulation approach to simulate peer’s state changes as well as attack behavior in the game. Particularly, in the

evaluation of secrecy-2, probability distributed for victim's position is estimated by using Monte Carlo approach. From the simulation results, we can see that SP2P has lowest secrecy-2, making it vulnerable to IE attacks. Both SAFE and PP2P have secrecy-2 close to what CS has, showing they have good mitigation to IE attacks. The choice between SAFE and PP2P mainly depends on the bandwidth and hardware availability of peers.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

Over this decade, MMOGs have achieved significant business success. Given the fact that rendering and network technologies are evolving, we can expect this success will continue. To be competitive and attract more and more subscribers, the game must technically provide extraordinary user experience such as fluent interaction and fair competition. Because of the limitation of C/S architecture, we foresee P2P architecture, which offers excellent scalability and low-latency interaction, may provide a solution for better user experience in the future. However, P2P architecture faces many challenges, and cheating is a major issue that blocks P2P from widely adopted in industry.

In this thesis, we particularly focus on IE attack on a specific game component, i.e., interest management (IM). The attack leads to the expose of victim's sensitive information (i.e., position) to the attacker, resulting in a unfair game. We have formalized this problem by defining the functionality of IM, the corresponding scheme (or protocol) and adversary model. Especially, We have specified properties such as correctness, secrecy (secrecy-1 and secrecy-2) and bandwidth usage to evaluate IM schemes.

To mitigate IE attacks, we have proposed two novel IM schemes. The first one is SAFE, an IM scheme that relies on privacy preserving protocol to obfuscate sensitive information in the SL update stage. Specifically, SAFE models the SL update as a

disjointness test of two private sets. It guarantees only the disjointness test result is revealed to the publisher. The other IM scheme we have proposed is PP2P which relies on the trusted block to protect sensitive information and operations. The trusted block is an execution environment that supports isolation, interaction and secure communication. We have also provided an implementation of the trusted block based on hardware virtualization and trusted computing technologies.

For protocols aiming to improve security, property verification and evaluation are very important. To assess properties such as correctness and secrecy-1, we have employed a formal verification approach. In particular, we have chosen model checking with CSP# as the modeling language. We have modeled each IM scheme in CSP# w.r.t. benign, HBC and malicious behavior, and specified correctness and secrecy-1 properties. The verification result shows that all IM schemes satisfy correctness but only CS and PP2P satisfy secrecy-1.

To evaluate quantitative properties such as secrecy-2 and bandwidth usage, we have employed a simulation approach. Specifically, the simulator executes a lite game with various IM schemes for testing. Monte Carlo based simulation is adopted to estimate the probability distribution of victim's position for calculating secrecy-2. The result shows both SAFE and PP2P have significantly better performance on secrecy-2 compared to SP2P. For comparison, SAFE relies on cryptographic algorithms that have no special requirements on hardware, but has relatively high bandwidth usage. On the contrary, PP2P requires hardware supports such as virtualization and trusted computing, but achieves better mitigation to IE attacks and consumes much less bandwidth.

In terms of academic contribution, two schemes SAFE and PP2P have been proposed and they have been proven effective on mitigating IE attacks. Especially, the trusted block used by PP2P is a generic approach for protecting sensitive information. This can be extended to cover more components besides IM. Using trusted blocks is possibly

a common solution to solve general cheating issues in P2P based games. The model checking approach gives a convincing example of how functional properties can be formally evaluated. Moreover, to the best of our knowledge, this thesis is the first work that introduces Shannon entropy as the quantitative measurement for the impact of IE attacks. This entropy-based evaluation approach offers a uniform indicator to compare among different IM schemes. In terms of practical contribution, our research targets on improving MMOG's security, which has direct benefits to the game industry. In addition, the IM schemes we have proposed include clear design and implementation solutions.

8.2 Future Work

This thesis provides a good start for research on analysis and mitigation of IE attacks. We summarize a few topics that can be studied in the future.

8.2.1 Improvement of Network Performance

Considering the user adoption, we predict IM schemes without special hardware requirements are more likely to be accepted by industry in near future. However, SAFE has high bandwidth consumption, which limits its adoption. Network performance of SAFE can be further investigated and improved. Especially, 1-out-of-N oblivious transfer [107] protocols could be potential candidates to replace the KM-1 used in the SAFE to reduce bandwidth usage. Moreover, we will evaluate secrecy-2 and bandwidth consumption with various sizes of AOI, ER and DR.

8.2.2 Lightweight Implementation of Trusted Block

Currently, PP2P relies on virtualization and trusted computing to build trusted blocks. However, these two technologies are costly and complicated for desktop-level applications such as IM. Lightweight approaches to implement trusted blocks should be investigated.

Recently, Intel has released an instruction extension called Software Guard Extensions (SGXs) [7, 65, 92], which allow applications to set up a private zone of code and data with much less dependencies. This may provide a lightweight solution to implement trusted blocks.

8.2.3 More Realistic Simulation

Current quantitative evaluation only simulates each peer's movement and the mobility model does not take into consideration what information the player has obtained. In real game, the attacker however utilizes obtained information for better tactics. If the attacker's reaction is simulated based on his/her obtained information, more realistic simulation results can be obtained. The improved simulation may provide better understanding of the impact of IE attacks.

Appendix A

Background

In this chapter, we provide background knowledge relevant to this thesis, particularly to SAFE and PP2P protocols, as well as their verification. The knowledge covers a number of research fields, including privacy preserving, trusted computing and hardware virtualization. We will investigate each of them in the following sections.

A.1 Privacy Preserving Protocol

In this section, we provide basic cryptographic background towards privacy preserving protocols, which has been used in the protocols to construct SAFE (Chapter 4). The purpose of privacy preserving protocol is for parties to compute a function jointly over their separate inputs while keeping these inputs private [138]. This functionality can be fulfilled by homomorphic encryption [51], which is a special form of public key encryption (a.k.a., asymmetric encryption) [78] that allows specific computation on private inputs without disclosing them. The following paragraphs will first introduce asymmetric encryption, followed by descriptions of homomorphic encryption.

A.1.1 Public Key Cryptography

The public key cryptography is a cryptographic approach which employs asymmetric key to achieve safety properties. In general, public key cryptography generates a pair of the

Figure A.1: Public key cryptography applications

secret key sk and the public key pk . As shown in their names, sk must be kept secret while pk can go public. Especially, equation

$$D_{pk}(E_{sk}(x)) = D_{sk}(E_{pk}(x)) = x$$

is always satisfied where E and D are two operations on plaintext and ciphertext with an asymmetric key as the parameter. RSA [111] and Elliptic Curve Cryptography (ECC) [96] are two commonly used asymmetric encryption algorithms. The length of input and output for E and D is denoted by l_{crypt} .

The common usages of public key cryptography are encryption and digital signature. Public key encryption mainly provides the confidentiality of the plaintext. As shown in Figure A.1.a, Alice encrypts plaintext x by the receiver Bob's public key, i.e., $y = E_{pk_{bob}}(x)$. When Bob receives the ciphertext y , his private key sk_{bob} can be used for decryption, i.e., $x = D_{sk_{bob}}(y)$. It guarantees that if Bob keeps sk secret, only he can recover plaintext from encrypted message.

Digital signature is the other application of public key cryptography for authentication and integrity, which is shown in Figure A.1.b. The typical situation is that Alice wishes

Figure A.2: Cryptographic hash function

to prove she is the genuine source of the message. Alice digitally signs the message msg by encrypting its hash value¹ with her private key, i.e., $sig = E_{sk_{alice}}(H(msg))$. The hash value is generated by cryptographic hash function, which will be explained soon later. By receiving msg and sig , Bob can verify its source via computing $digest' = D_{pk_{alice}}(sig)$ and comparing it with the result of $H(msg)$. Note that, this verification also checks the integrity of the message.

A.1.2 Cryptographic Hash Function

Cryptographic hash functions are commonly used in protocols. As shown in Figure A.2, a cryptographic function produces fixed-length bits from an arbitrary message. The result of the cryptographic hash function is also called message digest or hash value. Generally, hash function can be written in the form of

$$digest = H(msg)$$

where msg is a message and $digest$ is the hash value or message digest of msg . A practical cryptographic hash function must at least satisfy the following properties.

- *One-way function.* Given msg , it should be easy to compute $digest$ while given $digest$ only, it is computationally infeasible to recover msg .
- *Weak collision resistance.* Given msg_1 , it is computationally infeasible to find another msg_2 , where $msg_1 \neq msg_2$, such that $H(msg_1) = H(msg_2)$.

¹Signing on hash value rather than the original message is for better performance consideration.

- *Strong collision resistance.* It is computationally infeasible to find two distinct messages msg_1 and msg_2 , where $msg_1 \neq msg_2$, such that $H(msg_1) = H(msg_2)$.

SHA-1 [38] is a commonly used cryptographic hash algorithm. The function outputs hash values in 160 bits. In 2013, Marc Stevens [120] engineered an attack to SHA-1, which produces a hash collision with a complexity of $2^{57.5}$ operations. However, SHA-1 is still secure in the context of online games.

A.1.3 Public Key Infrastructure

As mentioned above, digital signature can be used for authentication. However, it does not address the problem of proving the actual identity of the entity who holds the key pair. The Public Key Infrastructure (PKI) is a systematical way to solve this problem [105].

The main purpose of PKI is to allow secure distribution and use of certificates. A certificate is the information referring to public key released by Certification Authority (CA), which takes responsibility for verifying the correctness of actual identity of the entity requesting for the certificate to be issued and ensures that the information contained within the certificate is correct and digitally signed. The PKI is usually built as a foundation for other security services. To make PKI scalable, CAs are built with hierarchy.

For example, Alice may request a certificate for email signature. To do this, she generates a pair of asymmetric keys and prepares evidence proving that she is who she states (e.g., by providing personal identification card). The request of the certificate, with public part of key as well as identification evidence will be sent to CA for validation². With certificate, she can sign her email by her secret key and issue it with her certificate. The receiver can thereby verify the signature by the public key included in the certificate and may further verify the certificate is faithfully issued from the CA.

²In practice, the verification is actually done by Registration Authority (RA) which proves the binding of identity and public key.

A.1.4 Homomorphic Encryption

Homomorphic encryption is a special form of asymmetric encryption that allows specific computation on private inputs without disclosing them [51]. The computation is carried out on ciphertext, which generates an encrypted result. When the result is decrypted, it matches the results which comes from the operation but performed on the plaintext.

In detail, suppose that we have a public key pk and a secret key sk , the homomorphic encryption scheme, for all $r_1, r_2 \in R$, $p_1, p_2 \in P$ and $c_1, c_2 \in C$, holds that

$$E_{pk}(r, p_1) = c_1, E_{pk}(r, p_2) = c_2$$

$$D_{sk}(c_1) = p_1, D_{sk}(c_2) = p_2$$

$$E_{pk}(r_1, p_1) \odot E_{pk}(r_2, p_2) = E_{pk}(r_1 \oplus r_2, p_1 + p_2)$$

where P , R and C are the plaintext space, the randomness space and the ciphertext space with parameter l_{crypt} . The binary operations $+$, $\oplus$ and $\odot$ are defined over P , R and C spaces. Especially, $\langle P, + \rangle$ and $\langle R, \oplus \rangle$ are the groups written additively and $\langle C, \odot \rangle$ is the group written multiplicatively. An informal explanation of above equations is that we can get the sum of two plaintexts by multiplying their corresponding ciphertexts.

A.2 Hardware Virtualization

Computer hardware virtualization is a technology to create virtual machines (VMs) on one physical platform. It allows users who run VMs to have the illusion that they own a complete computing platform. The PP2P scheme described in Chapter 5 has employed hardware virtualization for building the trusted block. The software managing VMs is called *hypervisor* or *virtual machine manager* [34].

A typical virtualization system relies on featured hardware technologies including processor virtualization, memory virtualization and I/O virtualization. These technologies

Figure A.3: Processor virtualization

enable the system to multiplex limited physical hardware resources to multiple virtual machines.

A.2.1 Processor Virtualization

Processor virtualization, or instruction virtualization can be achieved by different methods such as dynamic binary translation [12], paravirtualization [139] and full virtualization [130]. Among these methods, full virtualization is the most popular one in recent years. Full virtualization is a form of classical virtualization, or trap-and-emulate virtualization [104] where the processor operates in dual modes, i.e., host (privileged) and guest (deprivileged), which is shown in Figure A.3. The virtual machines that execute in the guest mode are monitored by hardware for specific architectural events. Upon such an event, the execution in the guest mode will be suspended and hypervisor will take over to handle the event in the host mode, which is called a *trap*. Alternatively, the guest program itself can deliberately invoke *hypercall* to trigger a trap, which is commonly used for requesting hypervisor services. After the event is handled, the hypervisor should *release* the control back to the VM that was trapped. To achieve processor virtualization in an efficient way, the processor must distinguish the execution mode and respond for different execution behavior, which requires special hardware logic. Most computers shipping since 2009 support hardware-assisted virtualization which provides sound performance for processor virtualization [4, 71].

Figure A.4: Memory virtualization

A.2.2 Memory Virtualization

Memory virtualization is to provide physical Random Access Memory (RAM) to different virtual machines. That is, it provides an illusion to each VM that it owns continuous physical memory space with the address starting from zero.

Different memory spaces are defined as follows. Host-Physical Address (HPA) space refers to the memory space in the view of physical chips. Guest-Physical Address (GPA) refers to the memory space in the view of the particular virtual machine. Guest-Virtual Address (GVA) refers to the memory space used by application's process context. Essential hardware such as Memory Management Unit (MMU) is responsible for GVA-to-GPA and GPA-to-HPA translations. Figure A.4 shows these transitions by arrows between different address spaces.

The translation is ruled by a data structure called memory paging table. The table has multiple entries where each indicates the mapping and access control information. While GVA-to-GPA transition is managed by the OS kernel in each VM, GPA-to-HPA translation is configured with a paging table assigned for each VM. The memory isolation between VMs can be achieved by setting their paging tables for non-overlapped HPA space. Otherwise, it is also possible to set up a shared memory region for inter-VM communication by mapping pages in GPA from different VMs to the same page in HPA.

A.2.3 Security Enhancement Based on Virtualization

The hardware-assisted virtualization has been widely utilized to build Cloud systems. But in fact, it can also be employed for building better security systems. Those systems rely on efficient isolation and management features provided by the hardware-assisted virtualization. Especially, these features are provided by hardware and thereby lead to small size of software dependencies.

Most security systems based on hardware-assisted virtualization rely on hypervisors to provide security features. SecVisor [116] is a tiny hypervisor to maintain kernel integrity. The solution is based on AMD platform to trap transitions between user space and kernel space. Necessary protections are applied to eliminate improper memory access before committing the transition. If combined with Nested Paging Technology (NPT) [5], the total Trusted Computing Base [83] is only 3526 Lines of Code (LOC). However, SecVisor requires modified OS to cooperate. BitVisor [117] is a pass-through hypervisor to particularly enhance disk I/O security, such as Advanced Technology Attachment (ATA) [124] device encryption. Particularly, it intercepts data transferring through ATA host controller and encrypts/decrypts it by Advanced Encryption Standard (AES) [101]. The TCB of BitVisor is around 20K LOC. TrustVisor [90] is another hypervisor aiming to protect the execution of sensitive code on legacy systems. It can execute security-sensitive code, namely Pieces of Application Logic (PAL), in an isolated environment from the legacy system. It also provides a reduced virtual Trusted Platform Module (TPM) [126] for each PAL to facilitate multiple PAL instances. TrustVisor's TCB is around 6K LOC. The PP2P scheme relies on a similar approach to provide isolation and communication functionalities of the trusted block.

A.3 Trusted Computing

In this section, we provide background knowledge about trusted computing [30], which has been used to construct trusted block in PP2P protocol (Chapter 5). Trusted computing is a technology promoted by the Trusted Computing Group (TCG) which defines trust as: *An entity can be trusted if it always behaves in the expected manner for the intended purpose* [2].

The core functionality of trusted computing is based on a secure chip, namely Trusted Platform Module (TPM) [126]. In general, TPM must be installed on the platform that requires trusted computing functionalities. Rather than limiting what software can execute on the platform, trusted computing measures critical software stack on the platform, which is usually called trusted computing base (TCB) [83], as evidence. This measured evidence can be further attested by a remote party to verify whether the platform behaves as expected. We will detail TPM chip, measurement, remote attestation and key generation which are key techniques to implement trusted block. Lastly, some known issues for TPM are discussed.

A.3.1 Trusted Platform Module

TPM is a secure chip connected to the motherboard via LPC bus [72]. As shown in Figure A.5, it incorporates a cryptographic processor to generate random numbers, RSA keys and perform encryption/decryption/signing/SHA-1 operations. TPM is designed to be tamper-resistant to protect internal data. In particular, Endorsement Key (EK) and Storage Root Key (SRK) are stored persistently in its internal non-volatile memory. Other data such as Platform Configuration Registers (PCRs), Attestation Identity Keys (AIKs) and loaded user keys for encryption/signing are kept in the volatile memory. TCG defines a set of TPM commands and data structures [126]. The main processor on the

Figure A.5: Trusted platform module

platform can invoke commands to interact with TPM for specific functionalities. Nowadays, TPM has been shipped to many desktop computers. BitLocker [93] is a commonly used application in Microsoft Windows, which employs TPM for disk encryption.

A.3.2 Measurement

As trust is determined by whether the behavior is expected, it is vital to have a mechanism to measure the behavior. By the definition, the behavior is measured by a trust chain of entities. For example, given e_i the entity currently executing in the platform, before it passes control to e_{i+1} that may affect the behavior we plan to measure, e_i is responsible to record e_{i+1} in TPM to extend the trust chain. Therefore, the trust chain recorded in TPM can reflect the platform configuration. A verifier with this information can determine whether the platform is configured as expected, or whether all entities in the invocation chain are expected.

To this end, TPM defines a set of PCRs internally. For TPM v1.2, it has a minimum of 24 PCRs. A PCR only accepts one operation $TPM_PCRExtend(i, x)$ which performs

$$pcr_{i_{new}} \leftarrow H_{SHA-1}(pcr_{i_{old}} || x)$$

The trusted chain is recorded by PCR. To extend the trust chain from e_i to e_{i+1} , e_i should execute $TPM_PCRExtend(i, H(e_{i+1}))$.

Especially, the very first entity that affects platform behavior is e_0 . The process of PCR initialization, measurement of e_0 and passing control to e_0 is called Root of Trust Measurement (RTM). In particular, Static Root of Trust Measurement (SRTM) defines e_0 as the BIOS program and is performed upon platform reboot. However, the trust chain of SRTM contains every entity from the BIOS, bootloader, OS kernel, drivers, libraries and so on. to the final applications, making it too long to be efficiently verified. To fix it, Dynamic Root of Trust Measurement (DRTM) is invented. DRTM allows the e_0 to be any program on the platform and it can be initiated at any time, which provides flexibility. It is important to note that DRTM requires specific hardware supports from main processor and chipset to ensure DRTM is atomically executed and e_0 will be the exclusive running program on the platform after DRTM. Both Intel and AMD release their DRTM implementations [6, 70].

A.3.3 Remote Attestation

As the platform configuration is measured by PCRs, it would be convenient to attest it to a remote party, or a verifier. The challenge is that TPM must convince the verifier that PCRs are reported from this particular TPM rather than another TPM or a piece of software code. To solve this problem, TPM defines a command $TPM_Quote(pcrs, aik)$ to generate a certificate containing PCR values. The certificate is signed by a specific TPM key, called Attestation Identity Key (AIK), which is dedicated for signing data inside TPM. An AIK can be generated by TPM through key creation command. The generated AIK is certified by EK, proving the AIK is generated by the TPM who owns the EK. This EK shipped with TPM has a manufacture-issued certificate to prove this EK is properly installed to a genuine TPM. As a result, the verifier needs to walk through

this PKI-based certificate hierarchy to believe that this received quotation is truly from the genuine TPM that it claims to. After that, the verifier can compare PCRs in the quotation with reference values to determine whether the platform is configured in an expected way.

We stress that remote attestation allows the verifier knows what was running on that platform when TPM quotation command was invoked. However, it does not guarantee the platform configuration is kept unchanged. It is possible that the verified platform has changed its configuration after the quotation is generated [21].

A.3.4 Key Generation and Distribution

TPM can also generate cryptographic keys. Especially, TPM can limit their usage to particular platform configurations, which is called *sealing*.

The command $TPM_CreateKey(type, pcrs, ref)$ internally creates a key pair and limits its usage to certain platform configuration illustrated by PCR set $pcrs$ with expected value ref . Specifically, the purpose of this key pair is defined by $type$ which can be *ENC* for encryption/decryption or *SIG* for signing/verification. The command returns its public key pk and a key blob $kb = E_{srk}(sk || type || pcrs || ref)$ that contains the secret key, key type and the sealing information. Note that the key blob is encrypted by SRK which is a symmetric key that never leaves TPM. To use this key, it must be firstly loaded into TPM by command $TPM_LoadKey(pk, kb)$ which returns a key handle kh for manipulating the key afterwards. The key can be loaded even if the PCRs do not match what it was sealed to.

To decrypt a message by a loaded key pair, one can invoke $TPM_Decrypt(kh, c)$ where kh is the key handle and c is the message in ciphertext. The TPM will firstly check if PCR set selected by $pcrs$ has matched the value specified by ref and key type $type$ is *ENC*. If these two conditions are both satisfied, the TPM will decrypt c by sk and

return the result of $D_{sk}(c)$. Similarly, command $TPM_Sign(kh, msg)$ can be invoked to sign message msg (usually a hash value of original message) by the key handled by kh .

TPM can certify generated keys by command $TPM_CertifyKey(kh, aik)$ where kh is the key handle and aik is the AIK to sign the certificate. This key certificate includes public key, key type and sealing information. It can be attested to a verifier so that he/she can believe this key can be used for decryption/signing only when the particular platform is configured in the way that sealing information claims.

A.3.5 Known Issues

One problem of TPM is about privacy and availability [137]. It is a challenge to maintain anonymity while providing evidence of trust. The EK uniquely identifies the endorser of the platform (at least details of the TPM), which violates anonymity. Even though AIK is used, the identity of TPM is trackable. To solve it, Direct Anonymous Attestation (DAA)[118] is introduced in TPM v1.2. It is a protocol that enables attestation while protecting the user's privacy. DAA involves three entities—TPM platform, DAA issuer and DAA verifier. The issuer is responsible to verify TPM platform and to issue DAA credential to the platform. In verification step, the platform proves that the credential to the verifier. The core technique of DAA is a zero-knowledge proof [54] to guarantee that the verifier can validate the credential without losing the platform's privacy. Moreover, the verification progress can be performed between TPM platform and the verifier, making the verification more available.

Another issue is on sharing TPM in the virtualization system [13]. Currently, TPM hardware is not designed to handle simultaneous commands. In general, all commands are forwarded to TPM via TCG Software Stack (TSS) [57], a TPM driver (service) in OS. It is not a problem until system applied virtualization. While remote attestation normally requires a genuine hardware TPM, concurrent accessing to TPM hardware by

different VMs is difficult, which will result in interfere issues. Simply emulating TPM functionalities cannot achieve its purpose. When application is supposed to verify a genuine physical TPM, virtualized TPM that lacks of EK certificate from vendor cannot be validated by the protocol [13]. The optimal solution is to adopt TPM 2.0 [8, 127] which has intentional considerations on virtualization.

A.4 Summary

In this chapter, techniques employed in this thesis have been detailed. Primitives in cryptography such as cryptographic hash function, PKI and homomorphic encryption have been covered (employed by SAFE in Chapter 4). Hardware virtualization and trusted computing which are the basics of PP2P scheme (see Chapter 5) have been discussed.

Appendix B

Source Code of Model Checking

B.1 im.csp

```
1. #import "IM";

2. enum {BENIGN, HBC, MAL};
3. enum {ALICE, BOB};

4. channel sub2pub 0;
5. channel pub2sub 0;

6. channel noermmsg 0;
7. channel nogsmmsg 0;

8. var beh = [BENIGN, BENIGN]; //Behavior of Alice and Bob

9. var < Im > im;
10. var insl[2] = [true, true];

11. var < Knowledge > k; //Alice's knowledge
12. var first_round = true; //Is the first round of the protocol

13. var output_ready = false;
14. var a_output; //actual output
15. var e_output; //expected output
16. Update_Expected_Output(pub, sub) =
17. update_expected_output{
18.   output_ready = false;
19.   if (im.isRel(new Gs(pub), new Aoi(sub)) == true) {
20.     e_output = new Gs(pub);
21.   } else {
22.     e_output = new Ud();
23.   }
24. } → Skip;
```

```
25. Update_CEr(pub, sub) =
26. ifa (im.isGen(im.getCEr(pub)) == false) {
27.   forged_cer_insl_true{im.setCEr(pub, true); } → Skip □
28.   forged_cer_insl_false{im.setCEr(pub, false); } → Skip
29. };

30. Det_InSl(pub, sub) =
31.   sub_in_pubsl{im.setInSl(pub, true); } → Skip □
32.   [im.getRel() == false]sub_notin_pubsl{im.setInSl(pub, false); } → Skip;

33. Det_Rel() =
34. (
35.   pubpos_in_subaoi{im.setRel(true); } → Skip □
36.   pubpos_notin_subaoi{im.setRel(false); } → Skip
37. );

38. Det_AoiInEr(pub, sub) =
39. (
40.   [!first_round]subaoi_in_suber{im.setAoiInEr(pub, true); } → Skip □
41.   subaoi_notin_suber{im.setAoiInEr(pub, false); } → Skip
42. );

43. Det_PosInDr(pub, sub) =
44. (
45.   [!first_round]subpos_in_subdr{im.setPosInDr(pub, true); } → Skip □
46.   subpos_notin_subdr{im.setPosInDr(pub, false); } → Skip
47. );

48. #define KnowsBobPos (k.knows(new Gs(BOB)) || k.knows(new Er(BOB)) ||
                        k.knows(new Dr(BOB)) || k.knows(new Pos(BOB)) ||
                        k.knows(new Aoi(BOB)));

49. #define NegCorrectness (output_ready && e_output! = a_output);
50. #define NegSecrecy1 (KnowsBobPos && e_output! = new Gs(BOB));

51. #include "cs.csp";
52. CS_Benign() = benign{beh[ALICE] = BENIGN; } → CS_Rounds();
53. CS_HBC() = hbc{beh[ALICE] = HBC; } → CS_Rounds();
54. CS_Mal() = mal{beh[ALICE] = MAL; } → CS_Rounds();
55. #assert CS_Benign reaches NegCorrectness;
56. #assert CS_HBC reaches NegCorrectness;
57. #assert CS_HBC reaches NegSecrecy1;
58. #assert CS_Mal reaches NegSecrecy1;
```

```
59. #include "sp2p.csp";
60. SP2P_Benign() = benign{beh[ALICE] = BENIGN; } → SP2P_Rounds();
61. SP2P_HBC() = hbc{beh[ALICE] = HBC; } → SP2P_Rounds();
62. SP2P_Mal() = mal{beh[ALICE] = MAL; } → SP2P_Rounds();
63. #assert SP2P_Benign reaches NegCorrectness;
64. #assert SP2P_HBC reaches NegCorrectness;
65. #assert SP2P_HBC reaches NegSecrecy1;
66. #assert SP2P_Mal reaches NegSecrecy1;

67. #include "safe.csp";
68. SAFE_Benign() = benign{beh[ALICE] = BENIGN; } → SAFE_Rounds();
69. SAFE_HBC() = hbc{beh[ALICE] = HBC; } → SAFE_Rounds();
70. SAFE_Mal() = mal{beh[ALICE] = MAL; } → SAFE_Rounds();
71. #assert SAFE_Benign reaches NegCorrectness;
72. #assert SAFE_HBC reaches NegCorrectness;
73. #assert SAFE_HBC reaches NegSecrecy1;
74. #assert SAFE_Mal reaches NegSecrecy1;

75. #include "pp2p.csp";
76. PP2P_Benign() = benign{beh[ALICE] = BENIGN; } → PP2P_Rounds();
77. PP2P_HBC() = hbc{beh[ALICE] = HBC; } → PP2P_Rounds();
78. PP2P_Mal() = mal{beh[ALICE] = MAL; } → PP2P_Rounds();
79. #assert PP2P_Benign reaches NegCorrectness;
80. #assert PP2P_HBC reaches NegCorrectness;
81. #assert PP2P_HBC reaches NegSecrecy1;
82. #assert PP2P_Mal reaches NegSecrecy1;
```

B.2 cs.csp

```
1. channel srv2sub 0;

2. CS(pub, sub) = init_scheme_ Gs{output_ready = false; }
                  → (CS_Srv(pub, sub) ||| CS_Sub(pub, sub));
3. CS_Srv(pub, sub) = CS_Srv_Benign(pub, sub);
4. CS_Sub(pub, sub) = (CS_Sub_BenignHBC(pub, sub) □
                      [beh[sub] == MAL] CS_Sub_Mal(pub, sub));

5. CS_Srv_Benign(pub, sub) =
6. ifa (im.isRel(new Gs(pub), new Aoi(sub)) == true) { //rel
7.   srv2sub!new Gs(pub) → Skip
8. } else {
9.   nogsmg!0 → Skip
10. };

11. CS_Sub_BenignHBC(pub, sub) =
12. (
13.   srv2sub?gsmg{
14.     if (beh[sub] == HBC) {k.addKnowledge(gsmg); }
15.   } →
16.   protocol_output{a_output = gsmg; output_ready = true; } →
17.   Skip
18. ) interrupt (
19.   nogsmg?0 →
20.   protocol_output{a_output = new Ud(); output_ready = true; } →
21.   Skip
22. );

23. CS_Sub_Mal(pub, sub) =
24. (
25.   srv2sub?gsmg{
26.     if (beh[sub] == HBC) {k.addKnowledge(gsmg); }
27.   } →
28.   ((
29.     protocol_output{a_output = gsmg; output_ready = true; } → Skip
30.   ) □
31.   (
32.     protocol_output{a_output = new Ud(); output_ready = true; } → Skip
33.   ))
34. ) interrupt (
35.   nogsmg?0 →
36.   protocol_output{a_output = new Ud(); output_ready = true; } →
37.   Skip
38. );
```

```
39. CS_Move() =  
40. Det_Rel();  
  
41. CS_Rounds() =  
42. CS_Move();  
43. aliceispub  $\rightarrow$  Update_Expected_Output(ALICE, BOB);  
44. CS(ALICE, BOB);  
45. bobispub  $\rightarrow$  Update_Expected_Output(BOB, ALICE);  
46. CS(BOB, ALICE);  
47. nextround{first_round = false; k.clear()}  $\rightarrow$   
48. CS_Rounds();
```

B.3 sp2p.csp

```
1. SP2P_Sub_BenignHBC_SlUpdate(pub, sub) =
2. ifa (im.getAoiInEr(pub) == false){ //updateer
3.   updateer{im.setAoiInEr(pub, true); } → //update local ER
4.   Det_InSl(pub, sub);
5.   sub2pub!new Er(sub) → Skip
6. } else {
7.   noermmsg!0 → Skip
8. };

9. SP2P_Pub_BenignHBC_SlUpdate(pub, sub) =
10. (
11.   sub2pub?ermmsg{
12.     if (beh[pub] == HBC) {k.addKnowledge(ermmsg); }
13.   } →
14.   ifa (im.isGen(ermmsg) == false) {
15.     cheat → Stop
16.   } else {
17.     updatecer{im.setCEr(pub, ermmsg); } → Skip
18.   }
19. ) interrupt (noermmsg?0 → Skip);
20. updatesl{var t = im.isInSl(new Pos(pub), im.getCEr(pub)); insl[pub] = t; } → Skip;

21. SP2P_Sub_BenignHBC_GsUpdate(pub, sub) =
22. (
23.   pub2sub?gsmsg{
24.     if (beh[sub] == HBC) {k.addKnowledge(gsmsg); }
25.   } →
26.   ifa (im.isGen(gsmsg) == false) {
27.     cheat → Stop
28.   } else {
29.     protocol_output{
30.       var rel = im.isRel(gsmsg, new Aoi(sub));
31.       if (rel) {a_output = gsmsg; } else {a_output = new Ud(); } //rel
32.       output_ready = true;
33.     } → Skip
34.   }
35. ) interrupt (nogsmmsg?0 → Skip);

36. SP2P_Pub_BenignHBC_GsUpdate(pub, sub) =
37. ifa (insl[pub] == true) { //insl
38.   pub2sub!new Gs(pub) → Skip
39. } else {
40.   nogsmmsg!0 → Skip
41. };

42. SP2P_Sub_BenignHBC(pub, sub) =
43. SP2P_Sub_BenignHBC_SlUpdate(pub, sub);
44. SP2P_Sub_BenignHBC_GsUpdate(pub, sub);
```



```

45. SP2P_Pub_BenignHBC(pub, sub) =
46. SP2P_Pub_BenignHBC_SlUpdate(pub, sub);
47. SP2P_Pub_BenignHBC_GsUpdate(pub, sub);

48. SP2P_Sub_Mal_SlUpdate(pub, sub) =
49. (
50.  sub2pub!new Er(sub) → Skip □
51.  noerrmsg!0 → Skip □
52.  sub2pub!new Er(sub, true) → Skip □
53.  sub2pub!new Er(sub, false) → Skip
54. );

55. SP2P_Pub_Mal_SlUpdate(pub, sub) =
56. (sub2pub?errmsg{k.addKnowledge(errmsg); } → Skip) interrupt (noerrmsg?0 → Skip);

57. SP2P_Sub_Mal_GsUpdate(pub, sub) =
58. (
59.  pub2sub?gsmg{k.addKnowledge(gsmg); } → Skip
60. ) interrupt (nogsmg?0 → Skip);

61. SP2P_Pub_Mal_GsUpdate(pub, sub) =
62. (
63.  pub2sub!new Gs(pub) → Skip □
64.  nogsmg!0 → Skip □
65.  pub2sub!new Gs(pub, true) → Skip □
66.  pub2sub!new Gs(pub, false) → Skip
67. );

68. SP2P_Sub_Mal(pub, sub) =
69. SP2P_Sub_Mal_SlUpdate(pub, sub);
70. SP2P_Sub_Mal_GsUpdate(pub, sub);

71. SP2P_Pub_Mal(pub, sub) =
72. SP2P_Pub_Mal_SlUpdate(pub, sub);
73. SP2P_Pub_Mal_GsUpdate(pub, sub);

74. SP2P_Sub(pub, sub) = (SP2P_Sub_BenignHBC(pub, sub) □
                          [beh[sub] == MAL] SP2P_Sub_Mal(pub, sub));
75. SP2P_Pub(pub, sub) = (SP2P_Pub_BenignHBC(pub, sub) □
                          [beh[pub] == MAL] SP2P_Pub_Mal(pub, sub));
76. SP2P(pub, sub) = init_scheme{output_ready = false; } →
                      (SP2P_Pub(pub, sub) ||| SP2P_Sub(pub, sub));

77. SP2P_Move_Asym(pub, sub) =
78. Det_InSl(pub, sub);
79. Update_CEr(pub, sub);
80. Det_AoiInEr(pub, sub);

```

```
81. SP2P_Move() =  
82. Det_Rel();  
83. SP2P_Move_Asym(ALICE, BOB);  
84. SP2P_Move_Asym(BOB, ALICE);  
85. SP2P_Rounds() =  
86. SP2P_Move();  
87. aliceispub  $\rightarrow$  Update_Expected_Output(ALICE, BOB);  
88. SP2P(ALICE, BOB);  
89. bobispub  $\rightarrow$  Update_Expected_Output(BOB, ALICE);  
90. SP2P(BOB, ALICE);  
91. nextround{first_round = false; k.clear()}  $\rightarrow$   
92. SP2P_Rounds();
```

B.4 safe.csp

```
1. enum {ERUPDATE, MSG1, MSG2};
2. channel noerupdate 0;
3. channel nodrupdate 0;

4. SAFE_Sub_KM1(pub, sub) =
5. pub2sub?MSG1 →
6. updateer{
7.   im.setAoiInEr(pub, true);      //updateer
8. } →
9. Det_InSl(pub, sub);
10. sub2pub!MSG2 → Skip;

11. SAFE_Pub_KM1(pub, sub) =
12. updatedr{
13.   im.setPosInDr(pub, true);      //updatedr
14. } →
15. Det_InSl(pub, sub);
16. pub2sub!MSG1 →
17. sub2pub?MSG2 →
18. updateinsl{
19.   if (im.isInSl(new Dr(pub), new Er(sub)) == true) //insl
20.     {var t = true; insl[pub] = t; }
21.   else
22.     {var t = false; insl[pub] = t; }
23. } → Skip;

24. SAFE_Sub_BenignHBC_SlUpdate(pub, sub) =
25. ifa (!im.getAoiInEr(pub)){ //erupdate signal
26.   sub2pub!ERUPDATE → SAFE_Sub_KM1(pub, sub)
27. } else {
28.   (
29.     noerupdate!0 →
30.     SAFE_Sub_KM1(pub, sub)
31.   ) interrupt (nodrupdate?0 → Skip)
32. };

33. SAFE_Pub_BenignHBC_SlUpdate(pub, sub) =
34. (
35.   sub2pub?ERUPDATE →
36.   SAFE_Pub_KM1(pub, sub)
37. ) interrupt noerupdate?0 →
38. (
39.   ifa (!im.getPosInDr(pub)){ //drupdate
40.     SAFE_Pub_KM1(pub, sub)
41.   } else {
42.     nodrupdate!0 → Skip
43.   }
44. );
```

```
45. SAFE_Sub_BenignHBC_GsUpdate(pub, sub) =
46. (
47.   pub2sub?gsmg{
48.     if (beh[sub] == HBC) {k.addKnowledge(gsmg); }
49.   } →
50.   ifa (im.isGen(gsmg) == false) {
51.     cheat → Stop
52.   } else {
53.     protocol_output{
54.       var rel = im.isRel(gsmg, new Aoi(sub));
55.       if (rel) {a_output = gsmg; } else {a_output = new Ud(); } //rel
56.       output_ready = true;
57.     } → Skip
58.   }
59. ) interrupt (nogsmg?0 → Skip);

60. SAFE_Pub_BenignHBC_GsUpdate(pub, sub) =
61. ifa (insl[pub] == true) { //insl
62.   pub2sub!new Gs(pub) → Skip
63. } else {
64.   nogsmg!0 → Skip
65. };

66. SAFE_Sub_BenignHBC(pub, sub) =
67. SAFE_Sub_BenignHBC_SlUpdate(pub, sub);
68. SAFE_Sub_BenignHBC_GsUpdate(pub, sub);

69. SAFE_Pub_BenignHBC(pub, sub) =
70. SAFE_Pub_BenignHBC_SlUpdate(pub, sub);
71. SAFE_Pub_BenignHBC_GsUpdate(pub, sub);

72. SAFE_Sub_Mal_SlUpdate(pub, sub) =
73. (
74.   sub2pub!ERUPDATE → SAFE_Sub_KM1(pub, sub) □
75.   noerupdate!0 →
76.   (
77.     SAFE_Sub_KM1(pub, sub)
78.   ) interrupt (nodrupdate?0 → Skip)
79. );

80. SAFE_Pub_Mal_SlUpdate(pub, sub) =
81. (
82.   sub2pub?ERUPDATE →
83.   SAFE_Pub_KM1(pub, sub)
84. ) interrupt (noerupdate?0 →
85. (
86.   SAFE_Pub_KM1(pub, sub) □
87.   nodrupdate!0 → Skip
88. )
89. );
```

```

90. SAFE_Sub_Mal_GsUpdate(pub, sub) =
91. (
92.  pub2sub?pubstate{k.addKnowledge(pubstate); } → Skip
93. ) interrupt (nogsmg?0 → Skip);

94. SAFE_Pub_Mal_GsUpdate(pub, sub) =
95. (
96.  pub2sub!new Gs(pub) → Skip □
97.  nogsmg!0 → Skip □
98.  pub2sub!new Gs(pub, true) → Skip □
99.  pub2sub!new Gs(pub, false) → Skip
100. );

101. SAFE_Sub_Mal(pub, sub) =
102. SAFE_Sub_Mal_SlUpdate(pub, sub);
103. SAFE_Sub_Mal_GsUpdate(pub, sub);

104. SAFE_Pub_Mal(pub, sub) =
105. SAFE_Pub_Mal_SlUpdate(pub, sub);
106. SAFE_Pub_Mal_GsUpdate(pub, sub);

107. SAFE_Sub(pub, sub) = (SAFE_Sub_BenignHBC(pub, sub) □
                        [beh[sub] == MAL] SAFE_Sub_Mal(pub, sub));
108. SAFE_Pub(pub, sub) = (SAFE_Pub_BenignHBC(pub, sub) □
                        [beh[pub] == MAL] SAFE_Pub_Mal(pub, sub));
109. SAFE(pub, sub) = init_scheme{output_ready = false; } →
                    (SAFE_Pub(pub, sub) ||| SAFE_Sub(pub, sub));

110. SAFE_Move_Asym(pub, sub) =
111. Det_AoiInEr(pub, sub);
112. Det_PosInDr(pub, sub);
113. Det_InSl(pub, sub);

114. SAFE_Move() =
115. Det_Rel();
116. SAFE_Move_Asym(ALICE, BOB);
117. SAFE_Move_Asym(BOB, ALICE);

118. SAFE_Rounds() =
119. SAFE_Move();
120. aliceispub → Update_Expected_Output(ALICE, BOB);
121. SAFE(ALICE, BOB);
122. bobispub → Update_Expected_Output(BOB, ALICE);
123. SAFE(BOB, ALICE);
124. nextround{first_round = false; k.clear(); } →
125. SAFE_Rounds();

```

B.5 pp2p.csp

```
1. enum {ALICE_TB_SIGN, ALICE_TB_ENC, BOB_TB_SIGN,
        BOB_TB_ENC, ALICE_SIGN, ALICE_ENC};
2. enum {PUBLIC, PRIVATE};

3. var pp2p_tb_sign = [new AKey(ALICE_TB_SIGN, PRIVATE),
                      new AKey(BOB_TB_SIGN, PRIVATE)];
4. var pp2p_tb_verify = [new AKey(BOB_TB_SIGN, PUBLIC),
                        new AKey(ALICE_TB_SIGN, PUBLIC)];
5. var pp2p_tb_dec = [new AKey(ALICE_TB_ENC, PRIVATE),
                     new AKey(BOB_TB_ENC, PRIVATE)];
6. var pp2p_tb_enc = [new AKey(BOB_TB_ENC, PUBLIC),
                     new AKey(ALICE_TB_ENC, PUBLIC)];

7. channel tb_genermsg 0;
8. channel tb_revealgs 0;
9. channel tb_updatecer 0;
10. channel tb_insl 0;
11. channel tb_gengsmsg 0;

12. channel finish 0;

13. PP2P_Sub_BenignHBC_SlUpdate(pub, sub) =
14.   ifa (im.getAoiInEr(pub) == false){ //updateer
15.     updateer{im.setAoiInEr(pub, true); } → //update local ER
16.     Det_InSl(pub, sub);
17.     tb_genermsg!pub.(new Er(sub)) →
18.     tb_genermsg?errmsg{
19.       if (beh[sub] == HBC) {k.addKnowledge(errmsg); }
20.     } →
21.     sub2pub!errmsg → Skip
22.   } else {
23.     noerrmsg!0 → Skip
24.   };

25. PP2P_Pub_BenignHBC_SlUpdate(pub, sub) =
26.   (
27.     (
28.       sub2pub?errmsg{
29.         if (beh[pub] == HBC) {k.addKnowledge(errmsg); }
30.       } →
31.       tb_updatecer!sub.errmsg →
32.       tb_updatecer?0 →
33.       Skip
34.     ) interrupt (noerrmsg?0 → Skip)
35.   );
36. tb_insl!sub.(new Pos(pub)) →
37. tb_insl?_insl{insl[pub] = _insl; } → Skip;
```

```

38. PP2P_Sub_BenignHBC_GsUpdate(pub, sub) =
39. (
40.   pub2sub?gsmg{
41.     if (beh[sub] == HBC) {k.addKnowledge(gsmg); }
42.   } →
43.   tb_revealgs!pub.gsmg.(new Aoi(sub)) →
44.   tb_revealgs?Gs{
45.     if (beh[sub] == HBC) {k.addKnowledge(Gs); }
46.   } →
47.   protocol_output{a_output = Gs; output_ready = true; } →
48.   Skip
49. ) interrupt (nogsmg?0 → Skip);

50. PP2P_Pub_BenignHBC_GsUpdate(pub, sub) =
51. ifa (insl[pub] == true) {                                     //insl
52.   tb_gengsmg!sub.new Gs(pub) →
53.   tb_gengsmg?gsmg{
54.     if (beh[pub] == HBC) {k.addKnowledge(gsmg); }
55.   } →
56.   pub2sub!gsmg → Skip
57. } else {
58.   nogsmg!0 → Skip
59. };

60. PP2P_Sub_BenignHBC(pub, sub) =
61. PP2P_Sub_BenignHBC_SlUpdate(pub, sub);
62. PP2P_Sub_BenignHBC_GsUpdate(pub, sub);

63. PP2P_Pub_BenignHBC(pub, sub) =
64. PP2P_Pub_BenignHBC_SlUpdate(pub, sub);
65. PP2P_Pub_BenignHBC_GsUpdate(pub, sub);
66. var < Gs > pp2p_cast_gs;
67. var < Er > pp2p_cast_cer;
68. PP2P_TB() =
69. (
70.   (
71.     atomic{
72.       tb_updatecer?sub.ermg{
73.         pp2p_cast_cer = ermg.unpack(pp2p_tb_verify[1 - sub], pp2p_tb_dec[1 - sub]); } →
74.         ifa (pp2p_cast_cer == new Error() || im.isGen(pp2p_cast_cer) == false) {
75.           cheat → Stop
76.         } else {
77.           tb_updatecer!0{im.setCEr(1 - sub, pp2p_cast_cer)} → Skip
78.         }
79.       }
80.     ) □

```

```
80. (
81.   atomic{
82.     tb_insl?sub.pos →
83.     ifa (im.isGen(pos) == false) {
84.       cheat → Stop
85.     } else {
86.       tb_insl!im.isInSl(pos, im.getCEr(1 - sub)) → Skip
87.     }
88.   }
89. ) □
90. (
91.   atomic{
92.     tb_gengsmg?sub.gs →
93.     tb_gengsmg!(new PackedMsg(gs, pp2p_tb_sign[1 - sub], pp2p_tb_enc[1 - sub])) → Skip
94.   }
95. ) □
96. (
97.   atomic{
98.     tb_genermsg?pub.er →
99.     tb_genermsg!(new PackedMsg(er, pp2p_tb_sign[1 - pub], pp2p_tb_enc[1 - pub])) → Skip
100.  }
101. ) □
102. (
103.   atomic{
104.     tb_revealgs?pub.gsmg.aoi{
105.       pp2p_cast_gs = gsmg.unpack(pp2p_tb_verify[1 - pub], pp2p_tb_dec[1 - pub]);
106.     } →
107.     ifa (pp2p_cast_gs == new Error() || im.isGen(pp2p_cast_gs) == false ||
          im.isGen(aoi) == false) {
108.       cheat → Stop
109.     } else ifa (im.isRel(pp2p_cast_gs, aoi) == true) {      //rel
110.       tb_revealgs!pp2p_cast_gs → Skip
111.     } else {
112.       tb_revealgs!new Ud() → Skip
113.     }
114.   }
115. )
116. );
117. PP2P_TB();
```



```
118. PP2P_Sub_Mal(pub, sub) =
119. (
120. (
121. (
122.   tb_genermsg!pub.(new Er(sub)) → Skip □
123.   tb_genermsg!pub.(new Er(sub, true)) → Skip □
124.   tb_genermsg!pub.(new Er(sub, false)) → Skip
125. );
126.   tb_genermsg?errmsg{k.addKnowledge(errmsg); } →
127.   sub2pub!errmsg → Skip
128. ) □
129. sub2pub!(new PackedMsg(new Er(sub), new AKey(ALICE_TB_SIGN, PRIVATE),
                               new AKey(ALICE_TB_ENC, PUBLIC))) → Skip □

130. noerrmsg!0 → Skip
131. );
132. (
133.   pub2sub?gsmmsg{k.addKnowledge(gsmmsg); } →
134.   (
135.     tb_revealgs!pub.gsmmsg.(new Aoi(sub)) → Skip □
136.     tb_revealgs!pub.gsmmsg.(new Aoi(sub, true)) → Skip □
137.     tb_revealgs!pub.gsmmsg.(new Aoi(sub, false)) → Skip
138.   );
139.   tb_revealgs?gs{k.addKnowledge(gs); } →
140.   protocol_output{a_output = gs; output_ready = true; } → Skip
141. ) interrupt (nogsmmsg?0 → Skip);

141. PP2P_Pub_Mal(pub, sub) =
142. (
143. (
144.   sub2pub?errmsg{k.addKnowledge(errmsg); } →
145.   tb_updatecer!sub.errmsg →
146.   tb_updatecer?0 → Skip
147. ) interrupt (noerrmsg?0 → Skip)
148. );
149. (
150. (
151. (
152.   tb_gengsmmsg!sub.(new Gs(pub)) → Skip □
153.   tb_gengsmmsg!sub.(new Gs(pub, true)) → Skip □
154.   tb_gengsmmsg!sub.(new Gs(pub, false)) → Skip
155. );
156.   tb_gengsmmsg?gsmmsg{k.addKnowledge(gsmmsg); } →
157.   pub2sub!gsmmsg → Skip
158. ) □
159. sub2pub!(new PackedMsg(new Er(pub), new AKey(ALICE_TB_SIGN, PRIVATE),
                               new AKey(ALICE_TB_ENC, PUBLIC))) → Skip □

160. nogsmmsg!0 → Skip
161. );
```

```
162. PP2P_Sub(pub, sub) = (PP2P_Sub_BenignHBC(pub, sub)  $\square$ 
    [beh[sub] == MAL] PP2P_Sub_Mal(pub, sub)) ;
163. PP2P_Pub(pub, sub) = (PP2P_Pub_BenignHBC(pub, sub)  $\square$ 
    [beh[pub] == MAL] PP2P_Pub_Mal(pub, sub)) ;
164. PP2P(pub, sub) = init_scheme{output_ready = false; }  $\rightarrow$ 
    (PP2P_Pub(pub, sub) ||| PP2P_Sub(pub, sub));

165. PP2P_Move_Asym(pub, sub) =
166. Det_InSl(pub, sub);
167. Update_CEr(pub, sub);
168. Det_AoiInEr(pub, sub);

169. PP2P_Move() =
170. Det_Rel();
171. PP2P_Move_Asym(ALICE, BOB);
172. PP2P_Move_Asym(BOB, ALICE);

173. PP2P_Rounds_Legacy() =
174. PP2P_Move();
175. alice_is_pub  $\rightarrow$  Update_Expected_Output(ALICE, BOB);
176. PP2P(ALICE, BOB);
177. alice_is_sub  $\rightarrow$  Update_Expected_Output(BOB, ALICE);
178. PP2P(BOB, ALICE);
179. nextround{first_round = false; k.clear()}  $\rightarrow$ 
180. PP2P_Rounds_Legacy();

181. PP2P_Rounds() = PP2P_Rounds_Legacy() ||| PP2P_TB();
```

Author's Publications

- Jianan Hao and Suiping Zhou. SAFE: A Secure Area of Interests Filter for P2P-based MMOGs. In *Proceedings of the 10th IEEE International Conference on Scalable Computing and Communications*, 2010
- Jianan Hao and Wentong Cai. Trusted Block as a Service: Towards Sensitive Applications on the Cloud. In *Proceedings of the 10th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, 2011
- Jianan Hao and Wentong Cai. Measuring Information Exposure Attacks on Interest Management. In *Proceedings of the 26th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulations*, 2012
- Jianan Hao, Yang Liu, Wentong Cai, Guangdong Bai and Jun Sun. vTRUST: A Formal Modeling and Verification Framework for Virtualization Systems. In *Proceedings of the 15th International Conference on Formal Engineering Methods*, 2013
- Guangdong Bai, Jianan Hao, Jianliang Wu, Yang Liu, Zhenkai Liang and Andrew Martin. TrustFound: Towards a Formal Foundation for Model Checking Trusted Computing Platforms. In *Proceedings of the 19th International Symposium on Formal Methods*, 2014

References

- [1] Process Analysis Toolkit. <http://www.comp.nus.edu.sg/%7Epat>.
- [2] Trusted Computing Group. <http://www.trustedcomputinggroup.org/>.
- [3] Dewan Tanvir Ahmed and Shervin Shirmohammadi. Uniform and non-uniform zoning for load balancing in virtual environments. In *Embedded and Multimedia Computing (EMC), 2010 5th International Conference on*, pages 1–6. IEEE, 2010.
- [4] AMD. AMD Virtualization. <http://www.amd.com/en-us/solutions/servers/virtualization>.
- [5] AMD. *AMD-VTM Nested Paging*, 1976. <http://developer.amd.com/wordpress/media/2012/10/NPT-WP-1%201-final-TM.pdf>.
- [6] AMD. *Secure Virtual Machine Architecture Reference Manual*, 2005. <http://www.mimuw.edu.pl/~vincent/lecture6/sources/amd-pacifica-specification.pdf>.
- [7] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. Innovative Technology for CPU Based Attestation and Sealing. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy, HASP'13*, 2013.
- [8] Will Arthur, David Challener, and Kenneth Goldman. Platform Configuration Registers. In *A Practical Guide to TPM 2.0*, pages 151–161. Apress, 2015.
- [9] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up Data in P2P Systems. *Communications of the ACM*, 46(2):43–48, Feb 2003.
- [10] Nathaniel Baughman and Brian Neil Levine. Cheat-Proof Payout for Centralized and Distributed Online Games. In *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM'01*, pages 104–113, 2001.

REFERENCES

- [11] Nathaniel Baughman, Marc Liberatore, and Brian Neil Levine. Cheat-Proof Payout for Centralized and Peer-to-Peer Gaming. *IEEE/ACM Transactions on Networking*, 15(1):1–13, Jan 2007.
- [12] Fabrice Bellard. QEMU, A Fast and Portable Dynamic Translator. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference, ATEC’05*, pages 41–41. USENIX Association, 2005.
- [13] Stefan Berger, Ramón Cáceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer, and Leendert van Doorn. vTPM: Virtualizing the Trusted Platform Module. In *Proceedings of the 15th conference on USENIX Security Symposium*, USENIX-SS’06. USENIX Association, 2006.
- [14] C.E.B. Bezerra, F.R. Cecin, and C.F.R. Geyer. A3: A Novel Interest Management Algorithm for Distributed Simulations of MMOGs. In *Proceedings of the 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, DSRT’08, pages 35–42, Oct 2008.
- [15] Ashwin Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: Supporting Scalable Multi-Attribute Range Queries. *SIGCOMM Computer Communication Review*, 34(4):353–366, Oct 2004.
- [16] Ashwin Bharambe, John R. Douceur, Jacob R. Lorch, Thomas Moscibroda, Jeffrey Pang, Srinivasan Seshan, and Xinyu Zhuang. Donnybrook: Enabling Large-Scale, High-Speed, Peer-to-Peer Games. In *Proceedings of the Conference on Computer Communications (SIGCOMM)*, pages 389–400. ACM, Aug 2008.
- [17] Ashwin Bharambe, Jeffrey Pang, and Seshan Seshan. Colyseus: A Distributed Architecture for Online Multiplayer Games. In *Proceedings of the 3rd Conference on Networked Systems Design & Implementation*, NSDI’06, 2006.
- [18] Blizzard. World of Warcraft: The Buring Crusade Continues Record-Breaking Sales Pace. <http://eu.blizzard.com/en-gb/company/press/pressreleases.html?id=10011312>, 2007.
- [19] Jean-Sébastien Boulanger, Jörg Kienzle, and Clark Verbrugge. Comparing Interest Management Algorithms for Massively Multiplayer Games. In *Proceedings of the 5th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames’06. ACM, 2006.
- [20] David Brandt. Networking and Scalability in EVE Online. https://wiki.eveonline.com/en/wiki/Fanfest_2005_Videos, 2005.
- [21] Sergey Bratus, Nihal DCunha, Evan Sparks, and Sean Smith. TOCTOU, Traps, and Trusted Computing. In *Trusted Computing-Challenges and Applications*, pages 14–32. Springer, 2008.

REFERENCES

- [22] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, MobiCom'98, pages 85–97. ACM, 1998.
- [23] Eliya Buyukkaya, Maha Abdallah, and Gwendal Simon. A Survey of Peer-to-Peer Overlay Approaches for Networked Virtual Environments. *Peer-to-Peer Networking and Applications*, 8(2):276–300, 2015.
- [24] Patrick Caldwell. Blizzard Bans 59,000 WOW Accounts. <http://www.gamespot.com/articles/blizzard-bans-59000-wow-accounts/1100-6154708/>, 2006.
- [25] Emanuele Carlini, Laura Ricci, and Massimo Coppola. Integrating Centralized and Peer-to-Peer Architectures to Support Interest Management in Massively Multiplayer Online Games. *Concurrency and Computation: Practice and Experience*, 27(13):3362–3382, 2015.
- [26] Emanuele Carlini, Luara Ricci, and Massimo Coppola. Reducing Server Load in MMOG via P2P Gossip. In *Proceedings of the 11th Annual Workshop on Network and Systems Support for Games*, NetGames'12, pages 1–2, Nov 2012.
- [27] C. Carlsson and O. Hagsand. DIVE: A Multi-User Virtual Reality System. In *IEEE Virtual Reality Annual International Symposium*, VRAIS'93, pages 394–400, Sep 1993.
- [28] C. Carter, A. El Rhalibi, and M. Merabti. A Survey of AoIM, Distribution and Communication in Peer-To-Peer Online Games. In *Proceedings of the 21st International Conference on Computer Communications and Networks*, ICCCN'12, pages 1–5, Jul 2012.
- [29] M. Castro, P. Druschel, A. M. Kermarrec, and A. I.T. Rowstron. Scribe: A Large-scale and Decentralized Application-Level Multicast Infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):1489–1499, Sep 2006.
- [30] David Challener, Kent Yoder, Ryan Catherman, David Safford, and Leendert Van Doorn. *A Practical Guide to Trusted Computing*. IBM Press, first edition, 2007.
- [31] Edmund Clarke. The Birth of Model Checking. In Orna Grumberg and Helmut Veith, editors, *25 Years of Model Checking*, Lecture Notes in Computer Science, pages 1–26. Springer Berlin Heidelberg, 2008.
- [32] Edmund Clarke and Jeannette Wing. Formal methods: State of the Art and Future Directions. *ACM Computing Surveys (CSUR)*, 28(4):626–643, 1996.

REFERENCES

- [33] Amy Beth Corman, Peter Schachte, and Vanessa Teague. A Secure Group Agreement (SGA) Protocol for Peer-to-Peer Applications. *International Conference on Advanced Information Networking and Applications Workshops*, 1:24–29, 2007.
- [34] R. J. Creasy. The Origin of the VM/370 Time-Sharing System. *IBM Journal of Research and Development*, 25(5):483–490, Sep 1981.
- [35] Eric Cronin, Burton Filstrup, and Sugih Jamin. Cheat-Proofing Dead Reckoned Multiplayer Games. In *International Conference on Application and Development of Computer Games*, 2003.
- [36] Steven Rak Daniel and Daniel J. Van Hook. Evaluation of Grid-Based Relevance Filtering for Multicast Group Assignment. In *Proceedings of the 14th Distributed Interactive Simulation*, 1996.
- [37] Eric DeBusschere and Mike McCambridge. Modern Game Console Exploitation. 2012. <https://www.cs.arizona.edu/~collberg/Teaching/466-566/2012/Resources/presentations/2012/topic1-final/report.pdf>.
- [38] D. Eastlake, 3rd and P. Jones. US Secure Hash Algorithm 1 (SHA1), 2001.
- [39] Orna Grumberg Edmund Clarke, Jr. and Doron Peled. *Model Checking*. MIT Press, 1999.
- [40] Inc Even Balance. PunkBuster Online Countermeasures. <http://www.evenbalance.com>.
- [41] Everquest.com. Everquest. <http://everquest.station.sony.com/>.
- [42] Lu Fan, Phil Trinder, and Hamish Taylor. Design Issues for Peer-to-Peer Massively Multiplayer Online Games. *International Journal of Advanced Media and Communication*, 4(2):108–125, Mar 2010.
- [43] Stefan Fiedler, Michael Wallner, and Michael Weber. A Communication Architecture for Massive Multiplayer Games. In *Proceedings of the 1st Workshop on Network and System Support for Games*, NetGames’02, pages 14–22. ACM, 2002.
- [44] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient Private Matching and Set Intersection. pages 1–19. Springer-Verlag, 2004.
- [45] Davide Frey, Jrme Royan, Romain Piegay, Anne-Marie Kermarrec, Fabrice Le Fessant, and Emmanuelle Anceaume. Solipsis: A Decentralized Architecture for Virtual Environments. In *Proceedings of the 1st International Workshop on Massively Multiuser Virtual Environments*, MMVE’08, 2008.

REFERENCES

- [46] S. Fujita. Load Balancing of P2P MMORPG Systems With Hierarchical Area-of-Interest Management. In *Proceedings of the 16th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, SNPD'15, pages 1–6, Jun 2015.
- [47] Thomas Funkhouser. RING: A Client-Server System for Multi-User Virtual Environments. In *Proceedings of the 9th Symposium on Interactive 3D Graphics*, I3D'95, pages 85–92. ACM, 1995.
- [48] Chris GauthierDickey, Daniel Zappala, Virginia Lo, and James Marr. Low latency and Cheat-Proof Event Ordering for Peer-to-Peer Games. In *Proceedings of the 14th international Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV'04, pages 134–139. ACM, 2004.
- [49] L. Gautier and C. Diot. Design and Evaluation of MiMaze, a Multi-Player Game on the Internet. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, ICMCS'98, pages 233–236, Jun 1998.
- [50] L. Gautier, C. Diot, and J. Kurose. End-to-End Transmission Control Mechanisms for Multiparty Interactive Applications on the Internet. In *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies*, INFOCOM'99, pages 1470–1479, Mar 1999.
- [51] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford, CA, USA, 2009. AAI3382729.
- [52] J.S. Gilmore and H.A. Engelbrecht. A Survey of State Persistency in Peer-to-Peer Massively Multiplayer Online Games. *IEEE Transactions on Parallel and Distributed Systems*, 23(5):818–834, May 2012.
- [53] Oded Goldreich. *Foundations of Cryptography: Volume 1*. Cambridge University Press, 2006.
- [54] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.
- [55] Marcin Gorawski and Karol Stachurski. A Secure Event Agreement (SEA) Protocol for Peer-to-Peer Games. In *Proceedings of the 1st International Conference on Availability, Reliability and Security*, ARES'06, pages 34–41. IEEE Computer Society, 2006.
- [56] Chris Greenhalgh, Jim Purbrick, and Dave Snowdon. Inside MASSIVE-3: Flexible Support for Data Consistency and World Structuring. In *Proceedings of the 3rd International Conference on Collaborative Virtual Environments*, CVE'00, pages 119–127. ACM, 2000.

REFERENCES

- [57] Trusted Computing Group. Trusted Computing Group Software Stack. http://www.trustedcomputinggroup.org/resources/tcg_software_stack_tss_specification.
- [58] Jianan Hao and Wentong Cai. Trusted Block as a Service: Towards Sensitive Applications on the Cloud. In *Proceedings of the 10th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 73–82. IEEE Computer Society, 2011.
- [59] Jianan Hao and Wentong Cai. Measuring information exposure attacks on interest management. In *Proceedings of the 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation*, PADS '12, pages 133–144, Washington, DC, USA, 2012. IEEE Computer Society.
- [60] Jianan Hao, Yang Liu, Wentong Cai, Guangdong Bai, and Jun Sun. vTRUST: A Formal Modeling and Verification Framework for Virtualization Systems. In Lindsay Groves and Jing Sun, editors, *Formal Methods and Software Engineering*, Lecture Notes in Computer Science, pages 329–346. Springer Berlin Heidelberg, 2013.
- [61] Jianan Hao and Suiping Zhou. SAFE: A Secure Area of Interests Filter for P2P-based MMOGs. In *Proceedings of The 10th IEEE International Conference on Scalable Computing and Communications*, ScalCom'10. IEEE Computer Society, 2010.
- [62] Martin Hellman. A Cryptanalytic Time-Memory Trade-Off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.
- [63] C. A. R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8):666–677, August 1978.
- [64] Charles Antony Richard Hoare. *Communicating Sequential Processes*, volume 178. Prentice-hall Englewood Cliffs, 1985.
- [65] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. Using Innovative Instructions to Create Trustworthy Software Solutions. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP'13, page 11. ACM, 2013.
- [66] Susan Hohenberger and Stephen Weis. Honest-verifier Private Disjointness Testing Without Random Oracles. In *Proceedings of the 6th International Conference on Privacy Enhancing Technologies*, PET'06, pages 277–294. Springer-Verlag, 2006.
- [67] Shun-Yun Hu, Jui-Fa Chen, and Tsu-Han Chen. VON: A Scalable Peer-to-Peer Network for Virtual Environments. *IEEE Network*, 20(4):22–31, July 2006.

REFERENCES

- [68] Takuji Iimura, Hiroaki Hazeyama, and Youki Kadobayashi. Zoned Federation of Game Servers: A Peer-to-Peer Approach to Scalable Multi-player Online Games. In *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames'04, pages 116–120. ACM, 2004.
- [69] MDY Industries. Glider. http://wow.gamepedia.com/Goblin_Glider, 2007.
- [70] Intel. *Intel Trusted Execution Technology Measured Launched Environment Developer's Guide*. <http://www.intel.com/content/dam/www/public/us/en/documents/guides/intel-txt-software-development-guide.pdf>.
- [71] Intel. *Intel[®] Virtualization Technology (Intel[®] VT)*. <http://www.intel.com/technology/virtualization/technology.htm>.
- [72] Intel. *Low Pin Count Interface Specification*. <http://www.intel.com/design/chipsets/industry/lpc.htm>.
- [73] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten Van Steen. Gossip-Based Peer Sampling. *ACM Transactions on Computer System*, 25(3), Aug 2007.
- [74] Jehn-Ruey Jiang, Yu-Li Huang, and Shun-Yun Hu. Scalable AOI-Cast for Peer-to-Peer Networked Virtual Environments. In *Proceedings of the 28th International Conference on Distributed Computing Systems Workshops*, ICDCS'08, pages 447–452, Jun 2008.
- [75] Aaron St. John and Brian Neil Levine. Supporting P2P Gaming When Players Have Heterogeneous Resources. In *Proceedings of the international workshop on Network and operating systems support for digital audio and video*, NOSSDAV'05, pages 1–6. ACM, 2005.
- [76] Patric Kabus and Alejandro P. Buchmann. Design of a Cheat-Resistant P2P Online Gaming System. In *Proceedings of the 2nd International Conference on Digital Interactive Media in Entertainment and Arts*, DIMEA'07, pages 113–120. ACM, 2007.
- [77] Patric Kabus, Wesley W. Terpstra, Mariano Cilia, and Alejandro P. Buchmann. Addressing Cheating in Distributed MMOGs. In *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames'05, pages 1–6. ACM, 2005.
- [78] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

REFERENCES

- [79] Aggelos Kiayias and Antonina Mitrofanova. Testing Disjointness of Private Datasets. In *Financial Cryptography and Data Security*, pages 109–124. Springer, 2005.
- [80] Lea Kissner and Dawn Song. Privacy-Preserving Set Operations. In *Advances in Cryptology—CRYPTO 2005*, pages 241–257. Springer, 2005.
- [81] Björn Knutsson, Honghui Lu, Wei Xu, and Bryan Hopkins. Peer-to-Peer Support for Massively Multiplayer Games. In *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies*, INFOCOM’04, Mar 2004.
- [82] D. Kushner. Engineering EverQuest: Online Gaming Demands Heavyweight Data Centers. *IEEE Spectrum*, 42(7):34–39, Jul 2005.
- [83] Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in Distributed Systems: Theory and Practice. *ACM Transactions on Computer System*, 10(4):265–310, Nov 1992.
- [84] Rodger Lea, Yasuaki Honda, Kouichi Matsuda, and Satoru Matsuda. Community Place: Architecture and Performance. In *Proceedings of the 2nd Symposium on Virtual Reality Modeling Language*, VRML’97, pages 41–50. ACM, 1997.
- [85] J. Lee. Wage Slaves. <http://www.1up.com/do/feature?cId=3141815>, 2005.
- [86] Kang Li, Shanshan Ding, Doug McCreary, and Steve Webb. Analysis of State Exposure Control to Prevent Cheating in Online Games. In *Proceedings of the 14th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV’04, pages 140–145. ACM, 2004.
- [87] Elvis S. Liu and Georgios K. Theodoropoulos. Interest Management for Distributed Virtual Environments: A Survey. *ACM Computing Survey*, 46(4):51:1–51:42, Mar 2014.
- [88] Tainchi Lu, Chungnan Lee, Wenyang Hsia, and Mingtang Lin. Supporting Large-Scale Distributed Simulation Using HLA. *ACM Transactions on Modeling and Computer Simulation*, 10(3):268–294, 2000.
- [89] Mammoth. Mammoth: The Massively Multiplayer Prototype. <http://mammoth.cs.mcgill.ca>.
- [90] Jonathan M. McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil Gligor, and Adrian Perrig. TrustVisor: Efficient TCB Reduction and Attestation. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2010.

REFERENCES

- [91] Jonathan M. McCune, Bryan Parno, Adrian Perrig, Michael K. Reiter, and Arvind Seshadri. How Low Can You Go? Recommendations for Hardware-Supported Minimal TCB Code Execution. In *Proceedings of the 2008 Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, March 2008.
- [92] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. Innovative Instructions and Software Model for Isolated Execution. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP'13, page 10, 2013.
- [93] Microsoft. BitLocker. <http://www.microsoft.com/windows/windows-7/features/bitlocker.aspx>.
- [94] Microsoft. Xbox. <http://www.xbox.com/>.
- [95] J.L. Miller and J. Crowcroft. The Near-Term Feasibility of P2P MMOG's. In *Proceedings of the 9th Annual Workshop on Network and Systems Support for Games*, NetGames'10, pages 1–6, Nov 2010.
- [96] Victor S Miller. Use of Elliptic Curves in Cryptography. In *Lecture Notes in Computer Sciences; 218 on Advances in Cryptology—CRYPTO 85*, pages 417–426. Springer-Verlag New York, Inc., 1986.
- [97] MMOGCHART. MMOGCHART.com. <http://www.mmogchart.com/>, 2008.
- [98] Katherine L. Morse, Lubomir Bic, and Michael Dillencourt. Interest Management in Large-Scale Virtual Environments. *Presence: Teleoperators and Virtual Environments*, 9(1):52–68, Nov 2000.
- [99] Christoph Neumann, Nicolas Prigent, Matteo Varvello, and Kyoungwon Suh. Challenges in Peer-to-Peer Gaming. *SIGCOMM Computer Communication Review*, 37(1):79–82, 2007.
- [100] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. LNCS. Springer, 2002.
- [101] National Institute of Standards and Technology. Advanced Encryption Standard (AES). <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>, 2001.
- [102] Ke Pan, Wentong Cai, Xueyan Tang, Suiping Zhou, and Stephen John Turner. A Hybrid Interest Management Mechanism for Peer-to-Peer Networked Virtual Environments. In *Proceedings of the 24th International Parallel & Distributed Processing Symposium*, pages 1–12, 2010.

REFERENCES

- [103] Joseph D. Pellegrino and Constantinos Dovrolis. Bandwidth Requirement and State Consistency in Three Multiplayer Game Architectures. In *Proceedings of the 2nd Workshop on Network and System Support for Games*, NetGames'03, pages 52–59. ACM, 2003.
- [104] Gerald J. Popek and Robert P. Goldberg. Formal Requirements for Virtualizable Third Generation Architectures. *Communications of the ACM*, 17:412–421, 1974.
- [105] Joel Weise-SunPSSM Global Security Practice. Public Key Infrastructure Overview.
- [106] Matthew Pritchard. How to Hurt the Hackers. pages 28–30, Jun 2000.
- [107] Michael O. Rabin. How To Exchange Secrets With Oblivious Transfer. Cryptology ePrint Archive, Report 2005/187, 1981. <http://eprint.iacr.org/>.
- [108] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. *SIGCOMM Computer Communication Review*, 31(4):161–172, Aug 2001.
- [109] Robert Rehner, Maribel Castro, and Alejandro Buchmann. nSense: Decentralized Interest Management in Higher Dimensions Through Mutual Notification. In *Proceedings of the 13th Annual Workshop on Network and Systems Support for Games*, NetGames'14, pages 1–3, Dec 2014.
- [110] Laura Ricci, Emanuele Carlini, Luca Genovali, and Massimo Coppola. AOI-Cast by Compass Routing in Delaunay Based DVE Overlays. In *Proceedings of International Conference on High Performance Computing and Simulation*, HPCS'11, pages 135–142, Jul 2011.
- [111] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
- [112] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In Rachid Guerraoui, editor, *Middleware 2001*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer Berlin Heidelberg, 2001.
- [113] Reuven Y Rubinstein and Dirk P Kroese. *Simulation and the Monte Carlo Method*. John Wiley & Sons, 2011.
- [114] Roi Saltzman and Adi Sharabani. Active Man in the Middle Attacks. *OWASP AU*, 2009.

REFERENCES

- [115] Arne Schmieg, Michael Stieler, Sebastian Jeckel, Patric Kabus, Bettina Kemme, and Alejandro Buchmann. pSense - Maintaining a Dynamic Localized Peer-to-Peer Structure for Position Based Multicast in Games. In *Proceedings of the 8th International Conference on Peer-to-Peer Computing*, P2P'08, pages 247–256, Sep 2008.
- [116] Arvind Seshadri, Mark Luk, Ning Qu, and Adrian Perrig. SecVisor: A Tiny Hypervisor to Provide Lifetime Kernel Code Integrity for Commodity OSES. *SIGOPS Operating System Review*, 41:335–350, Oct 2007.
- [117] Takahiro Shinagawa, Hideki Eiraku, Kouichi Tanimoto, Kazumasa Omote, Shoichi Hasegawa, Takashi Horie, Manabu Hirano, Kenichi Kourai, Yoshihiro Oyama, Eiji Kawai, Kenji Kono, Shigeru Chiba, Yasushi Shinjo, and Kazuhiko Kato. BitVisor: A Thin Hypervisor for Enforcing I/O Device Security. In *Proceedings of the ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE'09, pages 121–130. ACM, 2009.
- [118] Ben Smyth, Mark Ryan, and Liqun Chen. Direct Anonymous Attestation (DAA): Ensuring Privacy With Corrupt Administrators. In *Proceedings of the 4th European Conference on Security and Privacy in Ad-hoc and Sensor Networks*, ESAS'07, pages 218–231. Springer-Verlag, 2007.
- [119] GT Interactive Software. Quake. [https://en.wikipedia.org/wiki/Quake_\(video_game\)](https://en.wikipedia.org/wiki/Quake_(video_game)).
- [120] Marc Stevens. New Collision Attacks on SHA-1 Based on Optimal Joint Local-Collision Analysis. In Thomas Johansson and PhongQ. Nguyen, editors, *Advances in Cryptology EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 245–261. Springer Berlin Heidelberg, 2013.
- [121] Colin Stirling. Modal and Temporal Logics. 1991. <http://homepages.inf.ed.ac.uk/cps/uruguay1.pdf>.
- [122] Microsoft Studios. Gears of War. <https://gearsofwar.com>.
- [123] D. Stutzbach, R. Rejaie, and S. Sen. Characterizing Unstructured Overlay Topologies in Modern P2P File-Sharing Systems. *IEEE/ACM Transactions on Networking*, 16(2):267–280, Apr 2008.
- [124] T13. <http://www.t13.org>.
- [125] D. Terdiman. Virtual Good, Real Scams. <http://www.cnet.com/news/virtual-goods-real-scams/>, 2005.
- [126] Trusted Computing Group. *Trusted Platform Module Main Specification, Version 1.2, Revision 116*, 2011.

REFERENCES

- [127] Trusted Computing Group. *Trusted Platform Module, Family “2.0”, Revision 1.07*, 2014.
- [128] Valve. Valve Anti-Cheat System (VAC). https://support.steampowered.com/kb_article.php?ref=7849-Radz-6869, 2007.
- [129] M. Varvello, C. Diot, and E. Biersack. P2P Second Life: Experimental Validation Using Kad. In *IEEE INFOCOM 2009*, pages 1161–1169, April 2009.
- [130] VMware. Understanding Full Virtualization, Paravirtualization and Hardware Assist. http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf, 2007.
- [131] Georges Voronoi. Nouvelles Applications des Paramètres Continus à la Théorie des Formes Quadratiques. *Journal für die Reine und Angewandte Mathematik*, pages 97–102, 1908.
- [132] Steven Daniel Webb and Sieteng Soh. Cheating in Networked Computer Games: A Review. In *Proceedings of the 2nd International Conference on Digital Interactive Media in Entertainment and Arts*, DIMEA’07, pages 105–112. ACM, 2007.
- [133] Steven Daniel Webb and Sieteng Soh. A Survey on Network Game Cheats and P2P Solutions. *Australian Journal of Intelligent Information Processing Systems*, 9(4):34–43, 2008.
- [134] Steven Daniel Webb, Sieteng Soh, and William Lau. RACS: A Referee Anti-Cheat Scheme for P2P Gaming. In *Proceedings of the 17th International Workshop on Network and Operating Systems Support for Digital Audio & Video*, NOSSDAV’07, pages 37–42, 2007.
- [135] Steven Daniel Webb, Sieteng Soh, and Jerry Trahan. Secure Referee Selection for Fair and Responsive Peer-to-Peer Gaming. In *Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation*, PADS’08, pages 63–71. IEEE Computer Society, 2008.
- [136] Jeff Yan and Brian Randell. A Systematic Classification of Cheating in Online Games. In *Proceedings of the 4th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames’05, pages 1–9. ACM, 2005.
- [137] Yang Yang, Huanguo Zhang, Fan He, and Bo Zhang. Research and Security Analysis of Anonymous Identity Authentication in Trusted Computing. In *Proceedings of the 2009 International Conference on New Trends in Information and Service Science*, NISS’09, pages 409–412. IEEE Computer Society, 2009.
- [138] Andrew Chi-Chih Yao. How to Generate and Exchange Secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Oct 1986.

REFERENCES

- [139] Lamia Youseff, Rich Wolski, Brent Gorda, and Chandra Krintz. Paravirtualization for HPC Systems. In *Proceedings of the 2006 International Conference on Frontiers of High Performance Computing and Networking*, ISPA'06, pages 474–486. Springer-Verlag, 2006.
- [140] Anthony (Peiqun) Yu and Son T. Vuong. MOPAR: A Mobile Peer-to-Peer Overlay Architecture for Interest Management of Massively Multiplayer Online Games. In Wu chi Feng and Ketan Mayer-Patel, editors, *NOSSDAV*, pages 99–104. ACM, 2005.
- [141] B. Y. Zhao, Ling Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A Resilient Global-Scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, Sep 2006.
- [142] Li Zou, M.H. Ammar, and C. Diot. An Evaluation of Grouping Techniques for State Dissemination in Networked Multi-user Games. In *Proceedings of the 9th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 33–40, 2001.