Implementing mental poker without a Trusted Third Party

Dimitrios Mistriotis
dimitrios.mistriotis@kcl.ac.uk*

September 4, 2009

*you can also use dimitrismistriotis@gmail.com

Abstract

The aim of this project is to implement a security-related protocol using secure coding techniques and paradigms, mainly with respect to information flows.

The security protocol chosen is a specific instance of Mental Poker, a cryptographic protocol defined in 1979 with many interesting attributes. The reference language chosen is Java and its extension, Java with Information Flows, implemented in Cornell University.

This project, because of it's overambitious targets, did not reach its initially specified goal, which was a full implementation of a mental poker protocol in Java with Information Flows. This did not undermine the educational and research benefits of the whole procedure. Additionally this lays good foundations for further future research and implementations, since major portions of the protocol was implemented in Java as well as some steps towards an implementation in JIF.

Contents

1 Introduction **3**

2 Mental Poker **3**
- 2.1 Definition . . . 3
- 2.2 Assumptions . . . 4
- 2.3 Existing Work . . . 4
- 2.4 Data Structures . . . 4
  - 2.4.1 Distributed Notarization Chain . . . 4
  - 2.4.2 Card Vector Representation . . . 6
  - 2.4.3 Card Permutation Matrix . . . 7
  - 2.4.4 Delta and Epsilon Sets . . . 8
  - 2.4.5 Elgamal . . . 9
- 2.5 Algorithm Description . . . 10
  - 2.5.1 Introduction . . . 10
  - 2.5.2 Initialization . . . 11
  - 2.5.3 Card Draw . . . 11
- 2.6 Implementation of Mental Poker in Java . . . 12
  - 2.6.1 Initialization States . . . 13
  - 2.6.2 Card Draw States . . . 13

3 JIF **15**
- 3.1 Introduction to JIF . . . 15
- 3.2 Decentralized Label Model . . . 16
  - 3.2.1 Values . . . 16
  - 3.2.2 Principals . . . 16
  - 3.2.3 Labels . . . 16
  - 3.2.4 Relabeling . . . 17
- 3.3 Implicit and Explicit flows . . . 17
- 3.4 Decentralized Label Model in JIF . . . 18
  - 3.4.1 Example: Variable Declaration . . . 18

3.4.2 Example: Declassification . . . 18
3.4.3 Example: Array Handling . . . 18
3.4.4 Example: Classes, Method signatures and Exceptions . . . 19
3.5 Criticism of JIF . . . 19
3.6 Tools . . . 20
3.7 Learning JIF . . . 21

4 Mental Poker in JIF **22**
4.1 Introduction . . . 22
4.2 Methodology chosen . . . 22
4.3 Annotations . . . 22
4.4 Implementation . . . 24
4.4.1 Uplifting a Java class . . . 24
4.4.2 Porting a Java class . . . 25
4.4.3 Implementing a JIF class . . . 26

5 Evaluation **27**
5.1 Future Work . . . 28

6 Appendices **28**
6.1 Appendix A - Project Description . . . 28
6.2 Appendix B - Install JIF on ubuntu linux . . . 28
6.3 Appendix C - Directory structure of deliverable . . . 32
6.4 Appendix D - Java source code . . . 33
6.4.1 implementation_code/java_code/CardDrawState.java . . . 33
6.4.2 implementation_code/java_code/CardPermutationMatrix.java . . . 33
6.4.3 implementation_code/java_code/CardVectorRepresentation.java . . . 39
6.4.4 implementation_code/java_code/DNChain.java . . . 41
6.4.5 implementation_code/java_code/DataChainLink.java . . . 44
6.4.6 implementation_code/java_code/Deck.java . . . 49
6.4.7 implementation_code/java_code/DeltaEpsilonSet.java . . . 50
6.4.8 implementation_code/java_code/EcnryptedDeck.java . . . 51
6.4.9 implementation_code/java_code/ElGamal.java . . . 52
6.4.10 implementation_code/java_code/EncryptedCard.java . . . 53
6.4.11 implementation_code/java_code/EncryptedPermutationMatrix.java . . . 54
6.4.12 implementation_code/java_code/EncryptedVectorDeck.java . . . 57
6.4.13 implementation_code/java_code/InitializationState.java . . . 58
6.4.14 implementation_code/java_code/LcaseDeltaSet.java . . . 59
6.4.15 implementation_code/java_code/LcaseEpsilonSet.java . . . 60
6.4.16 implementation_code/java_code/MPElGamal.java . . . 60
6.4.17 implementation_code/java_code/MPEncryptedMessage.java . . . 65
6.4.18 implementation_code/java_code/MPGame.java . . . 66
6.4.19 implementation_code/java_code/Player.java . . . 67
6.4.20 implementation_code/java_code/Sha1Signature.java . . . 77
6.4.21 implementation_code/java_code/Signature.java . . . 78
6.4.22 implementation_code/java_code/UZeroGenerator.java . . . 79
6.4.23 implementation_code/java_code/VectorDeck.java . . . 81

6.5 Appendix E - JIF source code 82
6.5.1 implementation_code/jif_code/MPElGamal.jif 82
6.5.2 implementation_code/jif_code/MPEncryptedMessage.jif 83
6.5.3 implementation_code/jif_code/MPKeyPublic.jif 84
6.5.4 implementation_code/jif_code/PokerGame.jif 85

1 Introduction

According to the description of the project [6], the aim of this project is to produce a mental poker implementation, without the use of a Trusted Third Party and then use it as a basis for an implementation of a bigger security-based implementation in a secure language with respect to information flows. The language chosen for this project is Java with Information Flows (JIF). The aim of this project is to measure how easy or difficult it is for an MSc student to get accustomed with and evaluate the JIF concepts and Tools. There is also the difficulty of writing a useful application in such development environment.

2 Mental Poker

2.1 Definition

Mental Poker was originally introduced in the "Mathematical Gardener" by Rivest, Shamir and Adleman [19]. According to the original paper, "Mental Poker" is a game of poker played by two players, without a physical Deck, but with a use of a message exchange communication channel. At the beginning of the game, a random deal of the deck must be made. In the duration of the game players must know the cards that are their hand, but having no other information, such as which cards are on other players' hands or which are probable to be drawn next. Also no card should be selected twice. At the end of the game each player must e able to verify that no other players have cheated.

There are some important elements of the definition such as:

- The absence of a Trusted Third Party (TTP), which means that there is no external authority involved that will ensure that none of the players tried to cheat.
- The only way players can communicate is through the exchange of messages,
- These messages should have some specific properties, such as some of them being encrypted with an encryption algorithm which has some specific properties.

During the initial phases of the project many algorithms for playing mental poker extending the original one where identified, such as in [22] and [8]. The one chosen to be implemented for this dissertation is the: "Practical Mental Poker without a TTP Based on Homomorphic Encryption" [5]. We chose that particular for a number of reasons such as: It is the base for many other algorithms that actually derive from the one chosen, or offer alternatives to some of its properties. Therefore someone can implement the application if many other algorithms available, without much mental and programming effort. It has also been studied extensively for it's cryptographic properties and structure therefore it is a much safer choice. Finally a number of projects of implementing this particular algorithm with JIF, such as "Security-Typed Languages for Implementation of Cryptographic Protocols: A Case Study" [3] is available, offering a basis for comparison and measurement.

Apart from mental poker there are many other applications for these family of protocols such as other card games, games that require random numbers and random number generation from a predefined set of numbers, without the need for a Trusted Third Party orchestrating and coordinating the process. This makes them a very nice example of producing

an application where the study of individual information flows and it's security implication is being studied. This is the main reason why many JIF projects implement a Mental Poker algorithm.

2.2 Assumptions

In addition to the choice of the algorithm some further choices have to be made such as what is the most appropriate crypto-system. In section 3 of the algorithm description [5] is stated that:

> ... Permuting (i.e. shuffling) encrypted cards requires encryption to be homomorphic, so that the outcome of permuting and decrypting (i.e. opening) a card is the same that would be obtained if the card had been permuted without prior encryption (i.e. reversal) ...

After referencing the relevant literature, such as [18] or [12], as well as other implementations, the most appropriate algorithm was **Elgamal**. A proof of the algorithm's homomorphism can be found in: [14].

Elgamal is the prime example of homomorphic encryption as stated in [7], used in other relevant projects and had enough documentation and implementations available. An implication of this choice is the fact that the product of an Elgamal's encryption for a given cleartext number m, is a pair of numbers as encrypted message. Consequently different data structures should be used for encrypted messages, increasing significantly the size of code needed for implementation.

2.3 Existing Work

At the moment of writing this report there are some projects published with similar goals. Two of those are: Askarov's and Sabelfeld's implementation of Mental Poker in JIF [4] and "A Toolbox for Mental Card Games" [17] implemented in C++.

2.4 Data Structures

In order to implement the mental poker algorithm some data structures specific for this certain application are introduced:

- Distributed Notarization Chain (sect. 2.4.1)
- Card Vector Representation (sect. 2.4.2)
- Card Permutation Matrix (sect. 2.4.3)

2.4.1 Distributed Notarization Chain

This is an expansion of the concept of the Lamport Password Chains [11], where the expansion takes into account the existence of more than one entities, in this case the different mental poker players. DNCs are used in order to have each player to "sign" each of her actions so that if evaluation needs to occur a path can be constructed that appoints which player did what. In case of tampered or corrupted data, the player responsible for the event can be

pinpointed since the revealed data at the end of the game will give different signatures than the ones originally supplied. Every chain link m_k consists two elements: The Data Field D_k and the Chaining Value X_k.
Every Data Field further consists of three subfields:

- Timestamp (T_k),
- Concept (C_k) with the information that the link contains and
- Attributes (V_k) with the relevant to C_k information.

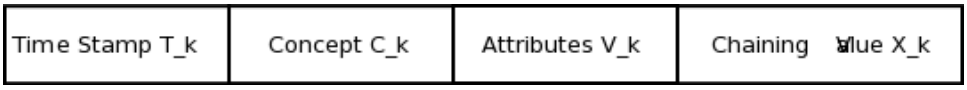

Figure 1: fields of a DNC

Addition of a new link to a chain can be done by each player individually, since the player needs to use only it's own X_{k-1} for signing. Therefore most operations in the mental poker protocol can be done in parallel. This is the reason why the players do not have to accept the links in the same order with one another: they can only watch and check each signature in comparison with the previous link.
When there is the need for players to synchronize a special "Chain contraction" link is being computed by a certain entity (which in mental poker's context will be the Croupier player). This is a mark that a milestone has been reached and the players can continue to an another part of the algorithm.

Illustrated in the following example:

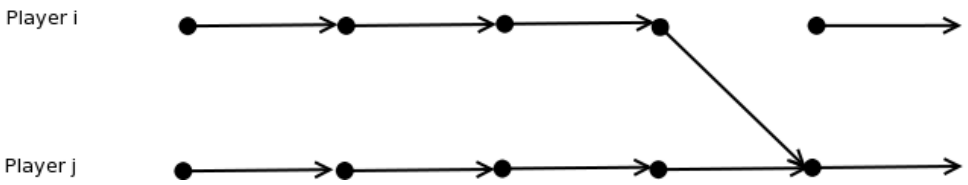

Figure 2: example of a DNC, here player j computes a concatenation link

The following Concepts are being used in this implementation:

Concept	Phase Used
z_i	Initialization (sect. 2.5.2)
Permutation Commitment	Initialization (sect. 2.5.2)
D-set	Initialization (sect. 2.5.2)
E-set	Initialization (sect. 2.5.2)
Encrypted Deck	Initialization (sect. 2.5.2)
First chain contraction	Initialization (sect. 2.5.2)
w_i Card vector representation	Card Draw (sect. 2.5.3)
$w_i^{'}$ Encrypted Card vector representation	Card Draw (sect. 2.5.3)

2.4.2 Card Vector Representation

Definition

Central in the mental poker algebra is the notion of the card and it's representation υ. In order to represent a card under this implementation, we need the total number of cards in the deck, **t**, and a prime number **z**. Each player p has her own prime number z_p. Each card is represented as a vector:
$\upsilon = (a_1, a_2, ..., a_t)$

In each vector υ exists an **i**, $1 \leq i \leq t$, for which there is only one a_i for which $a_i \pmod{z_p} \neq 0$. For that card vector, the value of the card represented is **i**.

Example

If $z = 7$ and $t = 4$ (player's prime number z is seven and deck has four cards), then the vector:
$\upsilon = (42, 21, 28, 50)$,
represents the card with value 4 ($i = 4$). That's because only for $a_4 = 50$, holds true that $a_4 \pmod{z_p} \neq 0$ ($a_4 \pmod{mod z_p} \neq 0 \Rightarrow 50 mod \pmod{7} \neq 0 \Rightarrow 1 \neq 0$). For all the other $i \neq 4$, $a_i \pmod{z_p} = 0$ ($42 \pmod{7} = 21 \pmod{7} = 28 \pmod{7} = 0$).

Implementation

For the implementation of the Card Vector Representation the following abbreviated java code was compiled:

```
public class CardVectorRepresentation {
    BigInteger[] uRepresentation; // Castella-Roca paper Definition
        1
    /* ... */
}
```

Note the following implementation details:
uRepresentation: is an array of Big Integers storing the actual a_i values.[1]

Constructors: A Card vector can be instantiated in three ways:

- Construct a card based on it's value sing the prime number z provided.
- By deserialization from a string, which might originate from network communication and
- as a the product of a card with a Permutation matrix (see Permutation Matrix Algebra, section 2.4.3).

[1]Since the protocol requires very big numbers with a large number of digits, "Big Integers" (in java instances of the *BigIntegerer* class) were used

2.4.3 Card Permutation Matrix

Definition

A deck of **t** cards is projected into a Permutation Matrix, which is a $t \cdot t$ matrix.

$$\Pi = \begin{pmatrix} \pi_{1,1} & \pi_{1,2} & ... & \pi_{1,t} \\ \pi_{2,1} & ... & ... & ... \\ ... & & & \\ \pi_{t,1} & \pi_{t,2} & ... & \pi_{t,t} \end{pmatrix}$$

where each i^{th} row of Π is a card $\pi(i)$.

Example

Suppose a permuted deck with four cards: $\pi = (4, 2, 3, 1)$ and that player's chosen prime number z is 7. A possible Permutation Matrix is the following:

$$\Pi = \begin{pmatrix} 21 & 14 & 28 & 51 \\ 42 & 19 & 7 & 35 \\ 36 & 49 & 42 & 14 \\ 35 & 28 & 44 & 7 \end{pmatrix}$$

The first row represents card 4, because the 4^{th} element is the only one for which $\pi_{1,4} \pmod z \neq 0$ or $51 \pmod 7 = 3 \neq 0$, therefore $\pi(1) = 4$. For the same reason $\pi(4) = 3$, since the 3^{rd} element of the fourth row, is the only one for which $44 \pmod 7 = 2 \neq 0$.

Permutation Matrix Algebra

The properties of the Permutation Matrix and the Vector representation of the cards by construction allow the matrix multiplication of a vector card, (an $q \times n$ array) representation with a permutation matrix (an $n \times n$ array). The result is a card with a different value.

If for example we calculate the card from the example 2.4.2 with the previous 2.4.3 permutation matrix we have:

$\upsilon \times \Pi =$

$$= (42, 21, 28, 50) \times \begin{pmatrix} 21 & 14 & 28 & 51 \\ 42 & 19 & 7 & 35 \\ 36 & 49 & 42 & 14 \\ 35 & 28 & 44 & 7 \end{pmatrix} == (4522, 3759, 4699, 3619)$$

The result Vector representation equals with the card with value 3, since the third element is the only one non-zero mod z_i, seven.

Equivalent Card Permutation Matrix

For each player the equivalent person matrix towards another player, is a Permutation Matrix that has the same card values, but is being calculated using the other player's prime number z. As in [5]:

$\Pi = \{\pi_{i,j}\}$ is equivalent to $\Pi^{'} = \{\pi^{'}_{i,j}\}$ **iff**
for each $i, j \in \{1, ..., numberofcardsindeck\}$:
$\pi_{i,j} \pmod{z} \neq 0 \iff \pi^{'}_{i,j} \pmod{z^{'}} \neq 0$ and $\pi_{i,j} \pmod{z} = 0 \iff \pi^{'}_{i,j} \pmod{z^{'}} = 0$

Equivalent Permutation Matrices are being used in the Card Draw phase of the protocol 2.5.3.

Implementation

Listing 1: Abbreviated listing of Card Permutation Matrix

```
class CardPermutationMatrix {
    private BigInteger[][] permutationMatrix = null;

    CardPermutationMatrix(VectorDeck inputVDeck) { /* ... */ }

    CardPermutationMatrix(Deck permuatedDeck, BigInteger zI) { /*
       ... */ }

    public CardPermutationMatrix getEquivalentPermutationMatrix(
       CardPermutationMatrix myMatrix, BigInteger myZ,
             BigInteger otherPlayerZ) { /* ... */ }

     public void modifyRowNonModuloZ(int row,BigInteger primeZ,
        SecureRandom rand) { /* ... */ }
    /* ... */
}
```

Again a two dimensional array of Big Integers is being used for internal representation, while all the actions described above on the matrix have their appropriate implementation.

2.4.4 Delta and Epsilon Sets

Definition

Two more special types of sets are used in Mental Poker with two instances each, δ, ϵ and their equivalents D and E. Those are defined for each player i as follows:

After a value s, such as $s > t$ is chosen, where t is the number of cards in the deck

δ is a set of s numbers such as for each number $n \in \delta$:

$n \pmod{z_i} = 0$, where z_i the z of player i.

ϵ is a set of s numbers that for number $n \in \epsilon$:

$n \pmod{z_i} \neq 0$, where z_i the z of player i.

D is a set generated from δ, with the key of player i, K_i in each d of D is the encrypted counterpart of δ:
for j element in δ, $d_j = E_{K_i}(\delta_j)$.

Similarly same for the E set (for j element in ϵ, $e_j = E_{K_i}(\epsilon j)$.)

Example

Assuming that player's i prime number is 7 and we have 5 cards in deck, the following is an example of δ_i:
$\delta_i = [56, 35, 63, 21, 28, 42]$

The set contains six numbers (one more than the deck size) all of them equal to 0 (mod 7). Similarly an ϵ set of the same player would be like:

$\epsilon i = [71, 9, 17, 30, 51, 37]$

Implementation

Listing 2: Abbreviated listing of δ set

```
class LcaseDeltaSet {
    protected BigInteger[] numCollection;

    LcaseDeltaSet(BigInteger primeZ) { /* ... */ }
}
```

As we see δ is represented as an array of Big Integers. ϵ class inherits it.

Listing 3: Abbreviated listing of Delta set

```
class DeltaEpsilonSet {
    MPEncryptedMessage[] BigDelta;

    DeltaEpsilonSet(LcaseDeltaSet lDelta, MPElGamal cryptoSystem) {
        /* ... */}
}
```

Similar to the δ implementation, D and E share the same code since both represent encrypted Big Integers.

2.4.5 Elgamal

Although not a data structure, the implementation of Elgamal's cryptosystem had to be customized. The rationale behind this decision has to do with the non-availability of the appropriate libraries in the JIF environment. The algorithm is an implementation from [18][2]

[2]pages 86 to 88

was adopted, taking into account secure coding techniques such as those described for coding in cryptography in [9]. Also some similar implementations were taken into account, one of them is [1]. A proof that Elgamal has the desired mathematical properties is here: `http://www.cs.ucla.edu/~rafail/TEACHING/WINTER-2005/L8/L8.ps`

Implementation

Listing 4: Abbreviated listing of custom Elgamal Implementation

```
public class MPElGamal {
    public MPKeyPublic getPublicKey(){ /* ... */ }

    public MPKeyPrivate getPrivateKey() { /* ... */ }

    public BigInteger getZPlayer() { /* ... */ }

    public MPEncryptedMessage encrypt(BigInteger message) { /* ...
       */ }

    public BigInteger decrypt(MPEncryptedMessage mpEnc) { /* ... */
        }

    public String signString(String message) { /* ... */ }

    public MPSignedInteger sign(byte[] hashedMessageByte) { /* ...
       */ }

    public static void main(String args[]) { /* ... */ }
}
```

The algorithm supports the encryption of a Big Integer into a pair of Big Integers as the algorithm implies, while the decryption is also supported. Additionally this is the place were the Elgamal capability to sign a message is utilized as well as the management of each player's private prime z_i. The main method is used for testing since a number is encrypted and decrypted in order to compare the results.

2.5 Algorithm Description

2.5.1 Introduction

The implementation of the chosen protocol [5] consists of two parts:

- Initialization (Sect. 2.5.2), and
- Card Draw (Sect. 2.5.3)

Players communicate **only** through messages (Sect. 2.1), which in this algorithm is Distributed Chain Links 2.4.1. In the following sections, the concepts are expanded and explained with additional information related to this project's nature.

2.5.2 Initialization

In the Initialization phase of the protocol players initialize some data structures which will be used later in the game, and broadcast the results of the computation to the other players.

When the result of the computation should be known to other players, then it is transmitted "as-is". In the case that the result should be kept secret, a hash-signature is transmitted as the attribute of the Distributed Notarization Chain.

More specifically:

1. Each player i generates an initial permutation of the Deck and a Secret Key K_i.
2. Each player i generates a large prime z_i, which is broadcasted to the rest of the players as a DNC-link.
3. A Card Permutation Matrix (Π_i) is generated out of the initial permutation from each player. Now a commitment of Π_i is broadcasted as a DNC-link. The player will also store the commitment for future evaluation
4. Each player will compute two sets: δ and ϵ (Sect. 2.4.4). Out of them the **D** and **E** sets will be produced. **D** set will be broadcasted as a DNC-link.
5. Then **E** will be broadcasted as a DNC-Link (Sect. 2.4.4).
6. A vector representation of the deck is being generated and then encrypted from each player.
7. The encrypted deck is being permuted and the next DNC-link containing the encrypted deck.
8. Finally a player selected acting as group croupier. In our implementation the first player is the croupier. computed the first DNC-link contraction and broadcasts it.

A croupier broadcast, signals the end of the initialization phase.

2.5.3 Card Draw

Card draw is being initiated whenever a player decides that a card should be drawn on his behalf [3].

Whenever a player decides to draw a card the following process [4] occurs:

1. a u_0 number is chosen, such as $0 \leq u_0 \leq t$ and u_0 has never been used before. Player requesting a card (PL_i) computes a w_0 representation of the card with value u_0 and broadcasts a w DNC-link.

[3] Card Draw is simplified since we run a simulated version of the game where two players can not decide that they want a card at the same time.

[4] the process has some minor modifications than that in the referenced algorithm, as stated here.

2. The first player (PL_1) receives the link and computes $w_1 = w_0.\Pi_1^{'}$, as described in section 2.4.3 where $\Pi_1^{'}$ is PL_1's equivalent permutation matrix of player that requested the card in the previous step. w_1 is broadcasted in a w DNC-link.

3. Each player PL_j before i in the list responds with a w_j DNC-link calculated as in the previous step.

4. When player i receives a w_{i-1} DNC-link does the following:

   - Permutes w_i using it's own Permutation Matrix (Π_i),
   - Modifies the m^{th} row of Π_i, where m the value of the received DNC-link
   - choses encrypted card $w_i^{'}$ which corresponds to w_i from the link
   - Broadcasts $w^{'}$ DNC-link

5. Players with order number bigger than the one requesting card, from the next one to the last one do the following:

   - Using the prime number z of player i (z_i) and the published key of the same player, computes an encrypted version of $\Pi_j^{'}$, named Π_j^c which involves as well the usage of the D and E sets computed in the initialization phase (the process here is exactly the same with [5] and therefore not mentioned here).
   - a $w_j^{'}$ is computed using the equivalent multiplication of $w_{j-1}^{'}$ with Π_j^c. The result is being broadcasted as a $w^{'}$ DNC-link.

6. When player i receives the broadcasted $w^{'}$ DNC-link from the last player, decrypts the link with it's private key and gets the value of the card. Card Draw finishes here.

2.6 Implementation of Mental Poker in Java

Mental poker was initially implemented abstracting the communication channels between the various entities, as suggested by Aslan Askarov and Andrei Sabelfeld in [3].

Initial versions of the algorithm included a communication mechanism that coordinated the different mental poker players, without the use of threads. The inability to use threads had as a consequence that no player could broadcast and receive messages [5] at the same time. In order to overcome this situation, a round robin policy on broadcasting was implemented (sect. 6.3). In later stages, this approach was abandoned in order to focus on the algorithmic aspects of Mental Poker, simplifying the development. A demonstration video and the source code are provided for future reference, along with the file deliverables.

In the current version each mental poker player is represented as a separate class with it's own private data. Communication between players is being handled by the entry point of the program. Since players can communicate only through DNCs, which additionally should be broadcasted, each players "next"-DNC is being requested and then passed as parameter to all players (including the one that broadcasted it). With this implementation we emulate a no-fault channel. In a real-world implementation this should be done by a dispatcher process from the host of the game, or in players reside inside a LAN (as in the implementation for the progress report, section 6.3), through broadcasted UDP packets.

[5] Messages were encapsulated in UDP-packets

Having the data structures compiled, the Java implementation looks more straightforward as the algorithm is "mapped" to Java equivalents using object orientated programming techniques. In our implementation each player entity is mapped to a Java class abstracting the communication mechanism. Also the players hold "fixed" positions, so they know their respective roles and can respond to appropriate DNC broadcast links.

Because each player must remember at every given moment in which state of the initialization or the card draw protocol is, a state machine approach was used, in order to formalize the code. Each player stores his/her current state in a enumerated structure.

2.6.1 Initialization States

"State-wise" Initialization is simple since players work in parallel, without co-ordination just broadcasting the results of their computation. Only deviation is at the end of the phase where one player, the Croupier, the first player in this implementation, will create a contraction link. So Croupier player has one additional state to pass from, before ending the initialization.

For initialization we have:

Listing 5: Initialization states

```
public enum InitializationState {
    START,
    ZIBROADCAST,
    PERMUTATIONMATRIXBCAST,
    UCASEDELTASETBCAST,
    BIGEPSILONSETBCAST,
    VECTORDECKBCAST,
    PLAYERCROUPIERBCAST,
    END
}
```

[6]

2.6.2 Card Draw States

Card Draw is different from Initialization because players have different roles in a draw. As described in the protocol in 2.5.3, we have:

1. The player requesting a card (assume PL_i),
2. Players before (PL_j, where $j < i$) and
3. Players after (PL_j, where $j > i$)

Player Requesting Card

After a player requests a Card, he/she has to wait for the appropriate DNC-link. At the same time the links received are stored for verification purposes, when the appropriate DNC-link is received then after the computations an encrypted vector card is broadcasted.

[6] Note that the last one (end) signals the transition to the card draw phase.

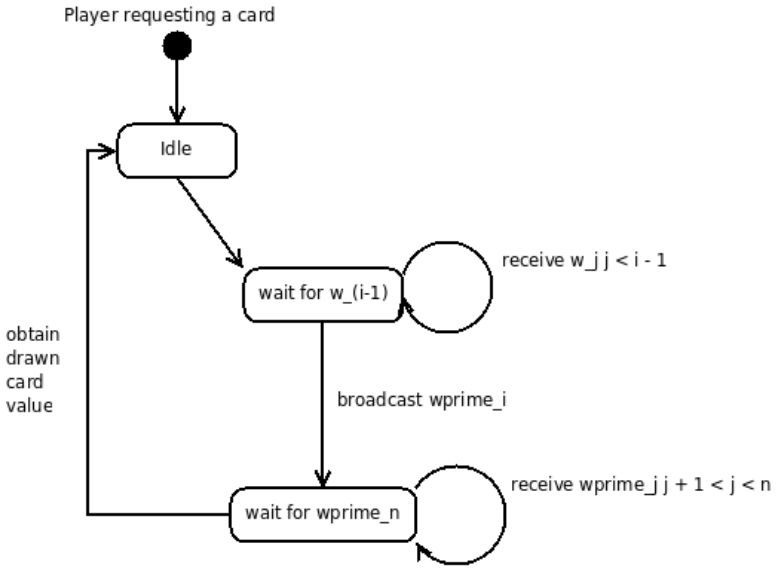

Figure 3: State Diagram of player drawing Card

After that the player goes again into recording DNC-link broadcasts until receiving one from the last player. This signals the end of Card Draw phase, so player returns back to being idle.

Player Before Player Requesting Card

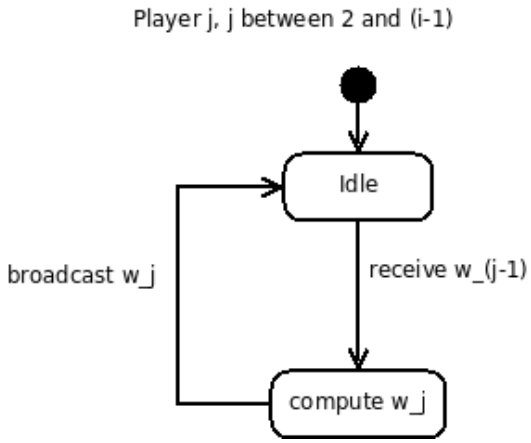

Figure 4: State Diagram of players before the one drawing a card

Players before the one requesting a card wait for their previous one to broadcast the appropriate DNC-link (Sect. 3) and then they respond with theirs. After that they return back to being idle.

Player After Player Requesting Card

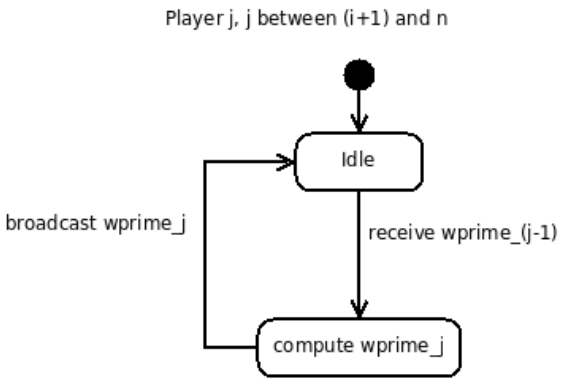

Figure 5: State Diagram of players after the one drawing a card

Players who are following the one requesting a card wait for their previous one to broadcast the appropriate DNC-link (sect. 5) and then they respond with theirs. After that they return back to being idle.

So this makes the Card Draw phase more complicated, since a player has to decide in which state to switch next. The states for Card Draw are the following:

Listing 6: Card Draw states

```
public enum CardDrawState {
    PLAYERIDLE,
    PLAYER1RCVW0,
    PLAYERJRCVJMINUSONE,
    PLAYERJRCVWJPRIMEMINUSONE,
    REQUESTCARDWAITWMINUSONE,
    CARDREQUESTWAITWPRIMEN,
}
```

3 JIF

3.1 Introduction to JIF

JIF [21] according to its home page is "a security-typed programming language that extends Java with support for information flow control and access control, enforced at both compile time and run time".

JIF is different than other similar programming language concepts in the way that the checks of information flow can occur both at compile time and at run time allowing a more flexible modeling of information flow. Theoretically this leads to simpler implementations which are also more flexible since they can solve more problems unsolved by other paradigms

which deploy static, only compile-time only flow checks. This is being achieved by implementing the Decentralized Label Model (sect. 3.2), discussed in the next section.

3.2 Decentralized Label Model

In the paper of Andrew C. Myers [2], JIF is described in contrast with other traditional models of information flow control. Briefly we have inside the code the concept of a Principal, who generally represents an entity or a user on which acts behalf of. Values (sect. 3.2.1) have attached Labels which define the access that different principals have to attached values. Each element as well of the Decentralized Label Model as well as the interactions among those elements are being described in the following sections.

3.2.1 Values

Values represent classes or basic types of the programming language. Additionally to the traditional model we have the "input channels" and the "output channels". "Input channels" represent points of entry of information in the program, and therefore can only be read. "Output channels" represent points where information is leaving the program, therefore can only be written.

3.2.2 Principals

Principals represent users or groups of users that interact within the system. Each value belongs to a principal, as we will see with the label concept below 3.2.3. The additional feature that JIF has in comparison with other paradigms is the capability of a principal to delegate its authority to others, with *acts-for* relationships. By default there are two principals available the "no-one" () and the "everyone" the former "acts-for" any principal, the latter is "acted-for" by any principal. These are used to ease the programming effort in some operations and introduce default behaviors.

3.2.3 Labels

Policies are implemented in JIF with labels. Each label constitutes of two sets of Principals, the reader set, those who can read-to a value, and the writer set, those who can write-to a value. The union of all principals in the reader set constitutes the effective reader set, which represents all principals that can read a value. While the intersection of the principals of the writer set, constitutes the principals that can write to a value.

These concepts allow two interesting actions in code and therefore in run-time level: the declassification and the restriction. With the declassification the reader set, therefore the effective readers, can be extended, allowing more principals to write to the value. While with a restriction action removes readers and/or adds owners. With these concepts, action policies can be implemented inside the code at run time as well as compile-time. While policies can only be implemented only at compile-time in other relevant paradigms, they can also be implemented at run-time here. The additional ways to define and implement a policy

constitute the additional capabilities of JIF and theoretically lead to more efficient and easy to read and evaluate code, in order to detect implicit and explicit flows.

3.2.4 Relabeling

A question rises on how declassification occurs internally. This is being done with the concept or re-labeling: a new value with a different label is being generated and the contents of the previous value are being copied to the new.

3.3 Implicit and Explicit flows

JIF also aims to solve the issue of explicit and implicit information flows:

- we have an **explicit flow** when information flows from a more restrictive to a less restrictive label,
- **implicit flows** occur when mixing variables of two different information classes and contents of one can be assumed from the values of the other.

Listing 7: Example of an explicit flow from Carnegie Mellon Information Security classes [10]

```
public class SecretMessages[principal alice, principal bob]
{
 String{alice:} aliceInstructions;
 String{bob:} bobInstructions;

 public SecretMessages(String{alice:} ai, String{bob:} bi) {
 aliceInstructions = ai;
 bobInstructions = bi;
 }
 public String{bob:} leak() {
 bobInstructions = aliceInstructions;
 return bobInstructions;
 }
}
```

Listing 8: Example of an implicit information flow from [2]

```
x = 0;
if b
{
    x = 1;
}
```

Here from the value of x, we can assume implicitly information about b.

3.4 Decentralized Label Model in JIF

In this section by presenting some working code or syntax examples, the way the theory behind the Decentralized Label Model will be illustrated and explained.

3.4.1 Example: Variable Declaration

First example is a variable declaration from [21]:

Listing 9: Variable declaration example

```
int {Alice: Bob} x;
```

Value x, an integer is owned by principal Alice. Principal Bob can read it.

3.4.2 Example: Declassification

Second example, is an example of a declassification (sect. 3.2.3) which leads to an output to screen[7]:

Listing 10: Declassification example

```
PrintStream[{}] output = declassify(runtime.stdout(new label{})
    , {});
if (output == null) return;

int{Alice:} iAlice = 3;
int aliceDec;
aliceDec = declassify(iAlice, {});
output.println("aliceDec: " + aliceDec);
```

Listing 10 is in a way both a "Hello World" code example as well as a declassification one. First an output channel is acquired label with the less restrictive label {}. Then the *iAlice* variable (named that way to identify an integer that belongs to principal Alice), is declassified to the less restrictive principal as well. Note that this is being done through relabeling (sect. 3.2.4). After that, the print-line call is legitimate and can be executed.

3.4.3 Example: Array Handling

Arrays are different in JIF due to the need of having two labels, one for the elements of the array and one for the array itself, as stated in [4]. This is being done in order to avoid the so-called "laundering attack".

[7] Code was created as part of this report, is publicly available and can be located at: `http://stackoverflow.com/questions/1037635/java-with-information-flows-output-to-screen`

Listing 11: This example is from the JIF-exercises [16]:

```
String {L}[]{L} larr;
```

Here *larr* is an array of Strings where both (array and elements) belong to label L.

3.4.4 Example: Classes, Method signatures and Exceptions

Apart from the method signatures, JIF code is generating a large number of Exceptions that should be handled inside the procedure or thrown. In this example we have a setter, which in a casual "setter" procedure.

As we see in this figure from [2]:

$$
\begin{array}{rcl}
\textit{procedure} & \rightarrow & \textit{id}\ \left[\ \textit{authority}\ \right] \\
 & & (\ \textit{arguments}\ ) \\
 & & \left[\ \textbf{returns}\ (\ \textit{id} : T_r\ \{\ L_r\ \}\ )\ \right] \\
 & & \textit{body}\ \textbf{end} \\
\textit{authority} & \rightarrow & \ll \textbf{caller} \gg \\
\textit{arguments} & \rightarrow & a_1 : T_1\left[\{L_1\}\right], \ldots, a_n : T_n\left[\{L_n\}\right]
\end{array}
$$

Figure 6: Procedure Syntax definition in JIF

Listing 12: The concept above is illustrated from an example from the JIF-exercises [16] as well:

```
public void setAt{L}(int {L} i, String {L} s):{L}
throws ArrayIndexOutOfBoundsException, ArrayStoreException,
   NullPointerException {
    /* Code Ommited */
}
```

3.5 Criticism of JIF

As with every programming concept or tool in the software sphere, JIF is not immune to criticism, both from outside and inside its programming community. Critique can be basically summarized into two positions:

- "JIF is too immature" as well as "JIF will never happen" [8] and
- "JIF provides a false sense of security" which usually pairs with "JIF is expensive".

For the rest of this section these two arguments will be briefly presented.

The "immaturity" argument has to do with the learning curve required to effectively write in JIF. This happens for two reasons, which together create a chicken and egg situation. Apart

[8] in the "IPv6 will never happen" concept

from programming with information flows is by definition hard, since first some additional concepts have to be mastered. Some, but not all, of them are the explicit and implicit flows, the concept of the side-effects, the high and the low level of the flow, the Principal algebra, etc. For this reason it is difficult to introduce these concepts into inexperienced or not too experienced programmers (they first need to know how to program and then on top program in some concepts), moreover amateurs or self-taught professionals. This more or less leads to a situation where in order for one to engage one needs to have an MSc-equivalent level and experience. This severely limits the potential JIF users, and the situation does not improve with the state of the current toolset which is inconsistent and has different stakeholders with different aims and desire for participation.

Hence if we had more people interested, there would be a bigger demand for quality tools and tighter integration. Similarly a better programming experience would attract more developers or generally people interested since the barrier of entry would be lower. The only way to break such a vicious circle is by large financial investments. Further discussion on this, however, is beyond the scope of this dissertation.

Another argument related to the maturity of the language, has to do with the fact that it took other such concepts more than 15 years to get established. The examples are numerous, the more obvious are version control, where many good tools were available in the early 1990's or Object Orientation, which was available many years before it became mainstream. Perhaps JIF is still in that incubation period, or does not yet have the critical mass it needs in the security software development community.

Regarding the "false sense.." argument, it is very common for a developer/project leader to fall into the fallacy of believing that a correct information flow modeling is a panacea that will solve all the project's security-related issues. Although this initially seems like an exaggeration, it is easy to fall into this category and ignore other aspects of a security related project, as the correct choice and implementation of crypto-systems or other secure coding approaches and practices. It is very easy to dwell into an information flow approach, something that JIF distribution does not discourage with the limited availability of ported programming libraries from the Java Runtime Environment. An obvious example is the custom implementation of a cryptographic algorithm from this project, something that can be considered a classic secure programming "mistake" (duplication of effort, use of an unproven algorithm, etc).

3.6 Tools

In this section a small review of tools available for creating programs in the JIF environment, at the date of writing this report is being presented.

JIF Distribution: The main distribution [21] offers the main compiler and the runtime environment. There are two issues here:

It is a source-only distributions and some time and familiarity with java compilation processes are needed in order to make JIF usable and some custom tweaks of the host system. The whole process is described in an Appendix of this report (appendix 6.2), since such a tutorial is not available. Different versions have different features and different level of maturity, considering bugs and capabilities. Therefore concerning the needs of the particular project, the choice must me made on the version that will be used. Also IDE (Eclipse) support is only

available into one of them.

JIF Eclipse [15]: JIF eclipse is a plugin for the popular eclipse platform. It assists the programmer with the creation and manipulation of programs, as well as pre-compilation analysis so that the programmer can immediately see an error before compilation, saving development time. It also offers suggestions which may help fixing such issues.

SIGGEN: From it's homepage [9], *"Siggen can automatically generate signatures for your Java and JIF files. It does not examine the bytecode to provide labels that match the security behavior of a library function"*

3.7 Learning JIF

For the course of this project the following JIF learning material was identified:

- **JIF's reference manual** [20]: Which has reference material on each version.
- **JIF Examples**: A set of examples like the well known Battleship game are included in the compiler package.
- **JIF Exercises** [16]: From the Chalmers University a series of exercises and tutorials demonstrated various concepts.
- **Blog entries**: Some relevant references on internet: `http://www.napes.co.uk/blog/decentralized-label-model/`
- **Other projects**: In the course of this project another MSc project was identified on implementing a "real-world" application in JIF, particularly a model of the French health care system [13].
- **JIF mailing list**: A low-activity mailing list by Cornell University.

Generally the learning material available falls into two categories: Firstly there is the material available inside academic papers submitted to conferences, publications etc. This generally emphasizes to the aspect that the author of the paper wants to for each relevant subject, without getting into the general context. Secondly there are some large scale implementations, such as Civitas [10], or Fabric [11].

The problem for the newcomer to the field is that there is no "middle ground" material available to aid in the course of getting the elementary concepts and combine them to a working application.

[9] http://www.cse.psu.edu/ dhking/siggen/

[10] Civitas: Toward a secure voting system. In Proc. IEEE Symposium on Security and Privacy, pages 354368, May 2008.

[11] "Fabric: A Platform for Secure Distributed Computation and Storage", Department of Computer Science, Cornell University

4 Mental Poker in JIF

4.1 Introduction

This project originally aimed at producing a Mental Poker implementation in Java with Information Flows. As it will be discussed later, this was extremely difficult to produce any usable result in the course of this project. In this section a summarization of the effort done will be presented as well as the material produced and the accompanying source code.

4.2 Methodology chosen

The methodology chosen to implement this project was inspired by [4], which is a paper based precisely on the experience of implementing the mental poker in JIF, as well as [13] which describes and suggests a similar paradigm. The first objective was the production of working code in java, so that the programmer can be accustomed with the protocol and the concepts around it. At some point in parallel, training in JIF takes place. After both of those have finished the programmer is capable of implementing the given algorithm in JIF. Different approaches lead either to very lengthy development times or just to project failure.

But even in this case, the workload can not be considered easy. Quoting [4]:

> ... The baseline implementation consumed around 60 man-hours of development work. The JIF implementation and distributed JIF implementation consumed 150 and 80 man-hours respectively, excluding the time to learn JIF. The case study indicates that although lifting Java code to JIF takes some experience to master, the security-typed result is not significantly distant from the original code. Further- more, we have developed patterns for secure programming (cf. Section 5) to make programming with security types clearer and more convenient. ...

What is missing from the quoted text is an estimation of "time required to learn JIF" and more importantly for this project, "how" can someone learn JIF. It appears from anecdotal evidence that usually before someone is assigned to a similar project, a MSc level semester course in secure programming with information flows is provided, usually with coursework and laboratory exercises in JIF. The process of self-teaching JIF, which was followed on this project in addition with the level of the educational available material (sect. 3.7), made the process harder which some times resulted to a trial-and-error approach with poor, at best, results.

4.3 Annotations

In the introductory paper to JIF [3] there were two motivational examples: a bank application and a medical research one. In these examples an annotation of information flow was used to illustrate the flow of information on the various principals.

In the examples provided the following annotations are being used:

Symbol	Usage
Circle	Principal within the system
Arrows	Information flow between principals
Square	Information flowing or Database
Double Circle	Trusted Agents that declassify information

In the first example, as shown in Sect. 4.3 we have the most simplified version of Mental Poker initialization phase among two players, Alice and Bob. Alice is assigned the role of the Croupier, therefore at the end of the initialization phase, she has to endorse Bob's contraction chains and produce a contraction link. That link has to be declassified and send back to Bob (which at this example represents all other players).

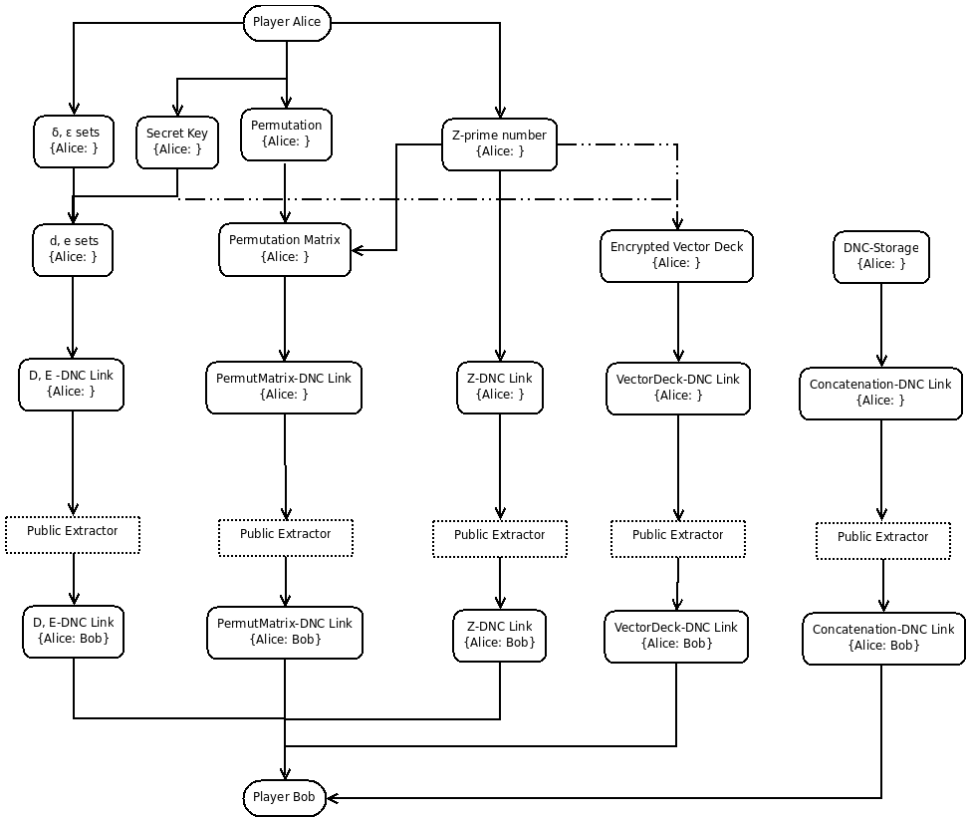

Figure 7: Mental Poker between Alice and Bob

The following step was to generalize this figure, into a multiplayer scenario, utilizing more JIF capabilities. Players mean of communication Distributed Notarization Chains (sect. 2.4.1), we wanted each player to have a coordinated picture of the game with each of the other mental poker players represented as a principal within the system. So each player j, would hold an array of Players i, where $i \neq j$. Each of them would hold their DNC-chain and would release what information is needed to current player j. In order for this to happen each DNC-received link, from an Input Channel (sect. 3.2.1) has to be identified, checked for validity and then elevated to be assigned to player's i DNC-chain. The only way to do this, as we see in figure (figure 4.3) is by having an input channel with the lowest principal, {*}, and then direct the input to an Extractor which will elevate it to higher levels. Then when the current player (Alice) needs a specific piece of information, it can be declassified and assigned.

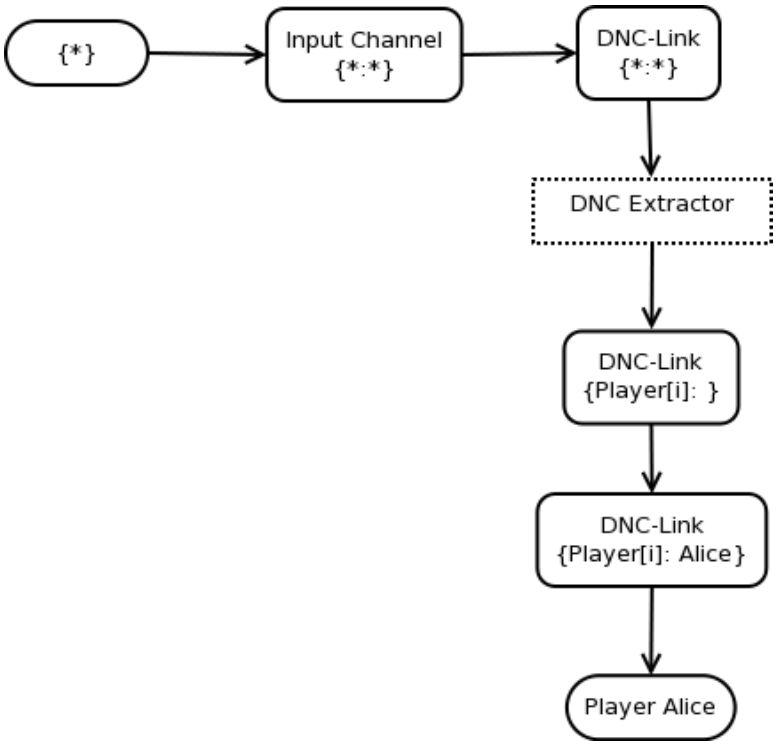

Figure 8: DNC link extraction

4.4 Implementation

Some examples of implementing parts of the Mental Poker protocol in JIF will be presented here.This will help the demonstration of the programming experience, the concepts and some language idiosyncrasies.

4.4.1 Uplifting a Java class

This is one way of porting an already existing java library to JIF. By supplying only the signatures, after compilation, the runtime environment can map the JIF signature to a Java one and execute it as desired. Please note that there is no check if the implementation maps to existing code and that some times the compilation process is rejected. Here we can see the "{this}" principal, which implies the current class status. Also the line:

private static int __JIF_SIG_OF_JAVA_CLASS$20030619 = 0;

Which according to JIF's creators, "is a hack to let the JIF compiler to know that the class provides a JIF signature for an existing Java class [20]".

Listing 13: Example of uplifting a class to JIF

```
package java.util;

public
class Random /*implements java.io.Serializable*/ {
```

```
    private static int __JIF_SIG_OF_JAVA_CLASS$20030619 = 0;

    /** use serialVersionUID from JDK 1.1 for interoperability */
    static final long serialVersionUID = 3905348978240129619L;

    public Random() {}

    public Random(long{this} seed) {

    }

    synchronized public native void setSeed(long{this} seed);

    public native void nextBytes(byte{this}[]{this} bytes);

    public native int{this} nextInt();

    public native int{this;n} nextInt(int n);

    public native long{this} nextLong();

    public native boolean{this} nextBoolean();

    public native float{this} nextFloat();

    public native double{this} nextDouble();

    synchronized public native double{this} nextGaussian();

}
```

4.4.2 Porting a Java class

In this section an example of porting a Java data structure to it's JIF equivalent will be provided along with some commendation on the programming experience. The class chosen is a simple data structure that holds three big integers as shown in Listing 14, and can only be ported in JIF as shown in Listing 15:

Listing 14: Java's Elgamal public key

```
class MPKeyPublic {
    public BigInteger p; //El Gamal's p, prime number
    public BigInteger g; //El Gamal's g, random number less than
    public BigInteger y; //El Gamal's y, y = (g^x)mod(p)
}
```

Listing 15: JIF Elgamal public key

```
import java.io.PrintStream;
import java.lang.Object;
import java.math.BigInteger;

class MPKeyPublic[label L] {

  private static int __JIF_SIG_OF_JAVA_CLASS$20030619 = 0;

  public BigInteger {L} p; //El Gamal's p, prime number
  public BigInteger {L} g; //El Gamal's g, random number less than
     p
  public BigInteger {L} y; //El Gamal's y, y = (g^x)mod(p)

  public void setP{L;newP}(BigInteger{L} newP): {L;newP} {
    this.p = newP;
  };

  public void setG{L;newG}(BigInteger{L} newG): {L;newG} {
    this.g = newG;
  };

  public void setY{L;newY}(BigInteger{L} newY): {L;newY} {
    this.y = newY;
  };
}
```

We can see programming elements such as the flagging of JIF for the class and the label: {L; newX}, which points that the higher and lower level of the caller must be the intersection of the label set (L) and the variable ($newP$ or $newG$ or $newY$). For this reason a "setter" function must be added, which was not as obvious as in the Java implementation.

4.4.3 Implementing a JIF class

This is an example in a so called "from scratch" programing approach, where the Elgamal algorithm had to be re-implemented since the underlying data structures had changed significantly. For simplicity the initialization is demonstrated.

Listing 16: The code is so different from the initial java implementation that it had to be completely rewritten.

```
...
MPElGamal{L; this}() {
    int{L} randomBL = this.RandomBitLength; //saves from side
       effect

    //initialization
```

```
    Random{L} rnd = new Random();
    publicKey  = new MPKeyPublic[L]{L}();
    privateKey = new MPKeyPrivate[L]{L}();
    //assignments
    BigInteger{L} biPublic;
    try {
        BigInteger{L} pubP =  new BigInteger{L}(randomBL, rnd);
        publicKey.setP(pubP);
        BigInteger{L} pubG = new BigInteger{L}(RandomBitLengthLess,
            rnd);
        publicKey.setG(pubG);
        BigInteger{L} privX =  new BigInteger{L}(
           RandomBitLengthLess, rnd);
        privateKey.setX(privX);

        BigInteger{L} pubY =  pubG.modPow(privX, pubP);
        publicKey.setY(pubY);

    } catch (java.lang.NullPointerException npExp) {
        //do nothing
    } catch (java.lang.IllegalArgumentException ilargExp) {
        //do nothing
    } catch (java.lang.ArithmeticException arExp) {
        //do nothing
    }
}
...
```

5 Evaluation

Although the desired result was not one was not reached per se, the experience was definitely positive, beneficial and educational in every aspect. During the course of this project there was exposure to diverse aspects of research and computer science concepts, such as software engineering, cryptography, mathematic, language-based security and their implementations. Additionally there was exposure to areas that are still under development which gave the experience of how concepts, ideas and implementations evolve in time.

Difficulties faced had to do with solving problems that in some times required knowledge of the domain in many aspects, such as programming, while in other cases there was need to invest in depth by reading the same references again and again, without an indication of a solution or knowing that the effort is pointed towards towards the right direction.

The project was over ambitious in it's targets and did not reach it's goals in full extent. However there has been a tremendous amount of work done, which can be utilized in further attempts as a foundation for future extensions. It also gave a fair amount of exposure on how research is being conducted and how outcomes and conclusion of a person's or team's research propagate to the rest of the academic community and influence future work.

5.1 Future Work

In case of further expansion to this project the following actions could be considered as logical next "steps":

The next logical step with respect to Mental Poker is the further implementation of the protocol, according to its specifications. Moreover, Code Verification could be implemented and incorporated in the existing project. Other directions could include Exceptional cases as a termination sequence when a player leaves the game [12], or when a connection is dropped. Finally a "real world" situation with formal network code instead of interprocess communication.

There are numerous approaches to extending the JIF aspect of this project. After some training in the language, the programming environment and the relevant JIF patterns. Additionally after familiarization with the tool-chain, a formal implementation can be produced from the mental poker code emphasizing on the aspects of Information Flow.

6 Appendices

6.1 Appendix A - Project Description

Java with Information Flow (JIF) is an extension to Java which provides security label annotations as part of the program and uses those labels to control information flow through a type system. Using such a type system it is possible to implement confidentiality and other security policies using the construction of the program. The aim of this project is to learn about JIF and writing programs in JIF, then to do a series of small programs building up to a significant one. A good possibility is an implementation of mental poker. This is a class of protocols in which there is on-going interest. using formal methods like state charts, B method and Perfect Developer to ensure critical properties A recent paper based on an MSc project implementation of such a mental poker protocol could form a useful jumping off point.

6.2 Appendix B - Install JIF on ubuntu linux

A. Install Sun's java

Due to issues with Sun's openness of Java packages, it is possible that a sun-java, which is the most preferable for JIF, is not installed by default on various linux distributions, including Ubuntu. This is about to change but not at the moment. If you are reading this a year or more after 2009, then probably this section is obsolete.

Following the instructions from url: `https://help.ubuntu.com/community/Java`, (which is to be merged in the server guide, so another location to check is: `https://help.ubuntu.com/8.10/serverguide/C/index.html`) and
`https://help.ubuntu.com/community/JavaInstallation`, type in a new terminal:
sudo apt-get install sun-java6-bin
for the JRE, which asks for accepting sun's terms and then:
sudo apt-get install sun-java6-jdk
for the SDK.

[12] Action which is not covered by this protocol

This should be OK from a clear installation of the OS, where there is no previous version of Java installed. If there is another version already installed then there should be a switch, described in the first url (sect. 6.2), which involves the following:

Open a Terminal window
Run
sudo update-java-alternatives -l
to see the current configuration and possibilities.
Run sudo update-java-alternatives -s XXXX to set the XXX java version as default. For Sun Java 6 this would be
sudo update-java-alternatives -s java-6-sun
Run **java -version**
to ensure that the correct version is being called.

JAVA_HOME variable

JAVA_HOME variable, which needs to exist for various scripts, at the moment of installation java was installed at: **/usr/lib/jvm/java-6-sun-1.6.0.10/**, so the /etc/bash.bashrc should be edited:
sudo gedit /etc/bash.bashrc
and the following line has to be added:
export JAVA_HOME=/usr/lib/jvm/java-6-sun-1.6.0.10/

In order to check for correctness use the following command:
echo $JAVA_HOME
in a new terminal window.

B. Install g++

Gnu g++ compiler is needed for compilation of JIF, therefore:
sudo apt-get install g++

C. Selection of JIF version

Normally someone might want to run the latest version of JIF available, in order to use the latest features, there is only one reason for doing otherwise: Eclipse-IDE.
The plugin for Eclipse, Jifclipse, which allows the use of the IDE in developing JIF applications, is unfortunately only compatible with JIF version 3. In the case that eclipse is not needed, any version can be installed. Either way compilation is exactly the same.

B. JIF compilation

JIF compilation scripts use ant for building mechanism, so if it is not installed:
sudo apt-get install ant

For the following installation steps, instructions in the "README" file inside the jif folder, are being followed. Before that it is useful do define JIF environment variable, in the same way as JAVA_HOME. Many developers prefer to install JIF in their own account and not system-wide, therefore the export command should be appended in /.bashrc. **gedit /.bashrc** and append directory's location. Example: *export JIF=/home/of/user/jif*
For system wide, edit /etc/bash.bashrc, the same file for JAVA_HOME and update accordingly.

For the rest of the installation this directory will be named **$JIF**, and the steps in the "README" file are followed:
In a new Terminal window
Run
cd $JIF
ant configure
which builds the runtime environment, for
the compiler:
ant

Now JIF runtime with the possible addition of compilation should have been installed. It is advised to compile and run the Battleship game provided with the language in order to check the installation.

E. JIFCLIPSE

If the eclipse platform is not installed: **sudo apt-get install eclipse**
After starting the IDE, instructions from
`http://siis.cse.psu.edu/jifclipse/`
can be followed.
Generally the installation is similar to one of a typical eclipse-plugin, with some additional actions of course.

1. Select: Help → Software Updates → Find and Install
2. Choose "Search for new features to install"
3. Click "New remote site"
4. **Name**: Jifclipse,
   URL: `http://siis.cse.psu.edu/jifclipse/update`
5. Click "Finish"
6. Select check box for "Jifclipse" and click "Next"
7. "Jifclipse feature 2.0.0" and read/accept license agreement
8. Click "Next" and "Finish"
9. Restart Eclipse

The only differences with the official instructions before was Eclipse's prompt to install an unsigned feature, followed by Eclipse prompting to restart itself.

The final step of Eclipse installation has to do with patching the JIF installation: [13]

1. Download the JIF compiler and follow the instructions provided to compile the JIF runtime (at least).
   Currently you must download JIF 3.0.0; we are working on integration of Jifclipse with more recent releases of JIF.
2. Download our updated jif.jar and build file: JIF updates for Jifclipse.
3. Unzip **jifclipse-aux.zip** into the JIF directory (the jifclipse-build.xml file should be in the main JIF directory, /usr/local/jif-3.0.0, e.g. and the jif.jar will be placed into /usr/local/jif-3.0.0/lib). You should be prompted whether you want to replace lib/jif.jar and you should answer "yes". Note: you MUST compile the JIF runtime before performing this update.
4. (If you needed to edit the build.xml file for JIF to add the headers include directory, the same edit will be required for the Jifclipse build file.)
5. Rebuild the JIF lib and SIG jars by running ant:
   ant -f jifclipse-build.xml
6. Start up Eclipse and use Project → New → JIF → JIF Project to create a new JIF Project
7. When creating a new JIF project, it should be configured to point to the base directory of the JIF compiler. See the Startup Tutorial for a demonstration.

In step 7 there are no instructions, since the how-to is in video only format.
In the eclipse: File → New → Project,
then select: "New Jif Project" and provide a name for the project, then click next. After this step there are two dialog boxes: **Base directory** for JIF compiler and **Path** to javac post compiler, where the appropriate directories must be set, in this case: */usr/local/jif-3.0.0* changed to */home/of/user/jif* and */usr/lib/jvm/java-1.5.0-gcj-4.3-1.5.0.0/jre/../bin/javac* to */usr/lib/jvm/java-6-sun-1.6.0.10/bin/javac*

[13] Note that from the quoted text (1) and (2) have already been done in steps (A) (B) and (D)

6.3 Appendix C - Directory structure of deliverable

List of files included in final report:

additional folder

jif-exercises: solutions for JIF exercises [16].

MentalPoker.bib: comprehensive list of all papers about mental poker, compiled in 2007.

december08_report: The preliminary report submitted on September 2008 which includes the code for the first steps of mental poker and a communication mechanism that avoids the use of threads .

implementation_code: Holds the code generated for this project:

java_code: Mental poker implementation in Java

jif_code: Mental poker implementation in JIF

6.4 Appendix D - Java source code

File listing of Java Files

6.4.1 implementation_code/java_code/CardDrawState.java

```
public enum CardDrawState {
    PLAYERIDLE, // (1) in card draw state diagrams
    PLAYER1RCVW0, // (2)
    PLAYERJRCVJMINUSONE, // (3)
    PLAYERJRCVWJPRIMEMINUSONE, // (4)
    REQUESTCARDWAITWMINUSONE, // (5)
    CARDREQUESTWAITWPRIMEN, // (6)
}
```

6.4.2 implementation_code/java_code/CardPermutationMatrix.java

```
import java.math.BigInteger;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;

class CardPermutationMatrix {
    private BigInteger[][] permutationMatrix = null;
    /*
     * Assumption: R1 and R2 of bit commitment protocol ("Bit
        Commitment Using one-way functions",
     * Handbook of Applied Cryptography p86-88), are BigInteger
     */
    private BigInteger commitmentR1 = null;
    private BigInteger commitmentR2 = null;
    private String commitment = null;

    final static int RandomNumberOfBits = 5;
    final static int CryptoError = -3; //Exit code

    /*note: CommitmentSeperator must be different than the one used
        in network transport
     */
    final static String CommitmentSeperator = "_";

    CardPermutationMatrix(VectorDeck inputVDeck) {
        int dimensionSize = Deck.numberOfCards;
```

```
        permutationMatrix = new BigInteger[Deck.numberOfCards + 1][
            Deck.numberOfCards + 1];
        //TODO: zero the 0 column and 0 row
        for (int row=0; row <= dimensionSize; row++) { //zero the 0
            th column
             permutationMatrix[0][row] = BigInteger.ZERO;
        }
        for (int column=0; column <= dimensionSize; column++) { //
            zero the 0th row
             permutationMatrix[column][0] = BigInteger.ZERO;
        }

        for (int column=1; column <= dimensionSize; column++) {
            //get Card at position of Column:
            CardVectorRepresentation currentCard = inputVDeck.
                getCardAtPosition(column);
            for (int row=1; row <= dimensionSize; row++) {//get
                each BigInteger of the card ansd assign it
                 permutationMatrix[column][row] = currentCard.
                     getVectorPosition(row);
            }
        }
    }

    /**
     *  Builds a card permutation on Deck permuatedDeck, using
        Players Z (zI)
     */
     CardPermutationMatrix(Deck permuatedDeck, BigInteger zI) {
         int numberOfCardsInDeck = Deck.numberOfCards;
         permutationMatrix = new BigInteger[numberOfCardsInDeck][
             numberOfCardsInDeck];
         SecureRandom secRnd = new SecureRandom();
         BigInteger randomBI = null;

         for (int row = 0; row < numberOfCardsInDeck; row++) {
             int currentCardValue = permuatedDeck.getCardAtPosition
                 (row);
             for (int column= 0; column < numberOfCardsInDeck;
                 column++) {
                  randomBI = new BigInteger(RandomNumberOfBits,
                      secRnd).abs();

                  BigInteger productRandomZi = zI.multiply(randomBI)
                      ;
                  permutationMatrix[row][column] = productRandomZi;
```

```
                if (currentCardValue == column) { //add number so
                    that it will not me a multiple of zI
                    randomBI = new BigInteger(RandomNumberOfBits,
                        secRnd).abs();
                    if (randomBI.compareTo(BigInteger.ZERO) == 0) {
                        randomBI = BigInteger.ONE;
                    } else { //number to add must be less than zI
                        randomBI = randomBI.mod(zI);
                    }
                    //System.out.println("random:" + randomBI + "
                        compare with zero:" + randomBI.compareTo(
                        BigInteger.ZERO) + " compare with zi" + zI.
                        compareTo(randomBI) + " Zi: " + zI);
                    permutationMatrix[row][column] =
                        permutationMatrix[row][column].add(randomBI)
                        ;
                }
            }
        }

    }


    BigInteger[][] getPermutationMatrix() {
        if (permutationMatrix != null) {
            return permutationMatrix;
        } else {
            return null;
        }
    }

    public CardPermutationMatrix getEquivalentPermutationMatrix(
        CardPermutationMatrix myMatrix, BigInteger myZ,
            BigInteger otherPlayerZ) {
        int numberOfCardsInDeck = Deck.numberOfCards;
        BigInteger[][] equivArray = new BigInteger[
            numberOfCardsInDeck][numberOfCardsInDeck];
        for (int row = 0; row < numberOfCardsInDeck; row++) {
            for (int column= 0; column < numberOfCardsInDeck;
                column++) {
                BigInteger currentCheck = this.permutationMatrix[
                    row][column];

                //make currentAssigned equals to zero mod
                    otherPlayerZ
                BigInteger currentAssigned;
                currentAssigned = currentCheck;
```

```
                currentAssigned = currentAssigned.subtract(
                    currentAssigned.mod(otherPlayerZ) );

                if (!currentCheck.mod(myZ).equals(BigInteger.ZERO)
                    ) {
                    BigInteger randomAdd = new BigInteger(
                        currentCheck.bitLength(), new SecureRandom
                        ());
                    randomAdd = randomAdd.mod(otherPlayerZ);
                    if (randomAdd.equals(BigInteger.ZERO)) {
                        randomAdd = BigInteger.ONE;
                    }
                    currentAssigned = currentAssigned.add(
                        randomAdd);
                }
                equivArray[row][column] = currentAssigned;
            }
        }


        CardPermutationMatrix resultPermMatrix = new
            CardPermutationMatrix(equivArray);
        return resultPermMatrix;
    }

    public void modifyRowNonModuloZ(int row,BigInteger primeZ,
        SecureRandom rand) {
        //some checks:
        if ( (row<1) || (row > Deck.numberOfCards) ) {
            System.err.println("error row out of range");
            System.exit(-4);
        }

        if (rand == null) {
            rand = new SecureRandom();
        }

        for (int i=1; i <= Deck.numberOfCards; i++) {
            BigInteger randomMultiple = new BigInteger(primeZ.
                bitLength() - 2, rand);
            randomMultiple = randomMultiple.multiply(primeZ);
            randomMultiple = randomMultiple.add(BigInteger.ONE);
            permutationMatrix[i][row] = randomMultiple;
        }
    }

    private CardPermutationMatrix(BigInteger[][] BIArray) {
```

```
        this.permutationMatrix = BIArray;

    }


    public String getCommitment() {
        if (commitment == null) {
            produceCommitment();
        }
        return commitment;
    }

    private void produceCommitment() {
        //Following the algorithm:
        //(1) Alice generates two random-bit strings R1 and R2:
        SecureRandom secRnd = new SecureRandom();
        commitmentR1 = new BigInteger(RandomNumberOfBits, secRnd).
           abs();
        commitmentR2 = new BigInteger(RandomNumberOfBits, secRnd).
           abs();
        //(2) Alice creates a message consisting of her random
           strings and
        //    the bit she wishes to commit to (it can actually be
           several bits)
        //    (R1, R2, b)
        byte[] messageR1 = commitmentR1.toByteArray();
        byte[] messageR2 = commitmentR2.toByteArray();
        byte[] messageb = null;
        //assumption, use a sha-1 instead of the whole data
           structure
        try {
           messageb  = Sha1Signature.getSHA1Byte(permutationMatrix
              .toString());
       } catch (NoSuchAlgorithmException e) {
           // TODO Auto-generated catch block
           e.printStackTrace();
           System.err.println("Sha1 Algorithm does not exist in
              this system");
           System.exit(CryptoError);
       }

        int messageLength = messageR1.length + messageR1.length +
           messageb.length;
        //concatenate: bit by bit
        byte[] message = new byte[messageLength];
        for (int i=0; i < messageR1.length; i++) {
            message[i] = messageR1[i];
```

```
        }
        int offset = messageR1.length;
        for (int i=0; i < messageR2.length; i++) {
            message[offset + i] = messageR1[i];
        }
        offset += messageR2.length;
        for (int i=0; i < messageb.length; i++) {
            message[offset + i] = messageb[i];
        }

        //(3) Alice computes the one-way function of the message
           and sends
        //the result, as well as one of the random strings, to Bob
           .
        //H(R1,R2,b), R1
        try {
            commitment = Sha1Signature.getSha1(message.toString())
               ;
        } catch (Exception exp) {
            exp.printStackTrace();
            System.err.println("Sha1 Algorithm does not exist in
               this system");
            System.exit(CryptoError);
        }
        commitment += CommitmentSeperator + messageR1.toString();
    }

    public String toString() {
        String result = "";
        int numberOfCardsInDeck = Deck.numberOfCards;
        for (int row = 0; row < numberOfCardsInDeck; row++) {
            for (int column= 0; column < numberOfCardsInDeck;
               column++) {
                result += permutationMatrix[row][column].toString
                   ();
                if ( (row != (numberOfCardsInDeck - 1)) || (column
                    != (numberOfCardsInDeck - 1)) ) {
                    result += ", ";
                }
            }
        }
        return result;
    }

    public static void main(String args[]) {
        //generate initial permutation and then a permutation
           matrix of it
```

```
        //Corresponds to step (e) of Initialization:
        //Build the card permutation matrix P_i corresponding to
           p_i using z_i
        BigInteger sillyPrime = new BigInteger("7"); //this should
           originate from player's key generation
        Deck initialPlayerDeck = new Deck();
        CardPermutationMatrix initialPermMatrix = new
           CardPermutationMatrix(initialPlayerDeck, sillyPrime);
        System.out.println( initialPermMatrix.toString() );
        //System.out.println( "Signature" + Signature.getSha1(
           initialPermMatrix.toString()) );
        System.out.println( "Commitment: " + initialPermMatrix.
           getCommitment() );

    }
}
```

6.4.3 implementation_code/java_code/CardVectorRepresentation.java

```
import java.math.BigInteger;
import java.security.*;

/*
 * First in (m) of initialization algorithm:
 * "... generate the vector representation of the t cards in Deck.
    ..."
 * note: first one is zero
 */

public class CardVectorRepresentation {
    BigInteger[] uRepresentation; // Castella-Roca paper Definition
        1
    int totalNumOfCards = -1;

    CardVectorRepresentation(int cardValue, int totalNumOfCards,
       BigInteger primeNumberZ, SecureRandom sec) {
        if (sec == null) {
            sec = new SecureRandom();
        }
        this.totalNumOfCards = totalNumOfCards;

        uRepresentation = new BigInteger[totalNumOfCards + 1];
        /* populate the array with elements equal zero (mod
           primeNumberZ)
```

```
         * and then add to the i-th element, so that the value of
           the card will be i
         */
        int numBits = primeNumberZ.bitLength() - 2; //order less
        uRepresentation[0] = null;
        for (int i = 1; i <= totalNumOfCards; i++) {
            BigInteger multiplier = new BigInteger(numBits, sec);
            uRepresentation[i] = multiplier.multiply(primeNumberZ);
            //multiple therefore equals zero mod primeNumberZ
        }

        BigInteger addition = new BigInteger(numBits, sec);
        addition = addition.mod(primeNumberZ.subtract(BigInteger.
           ONE));
        if (addition.equals(BigInteger.ZERO)) {
            addition = BigInteger.ONE;
        }
        uRepresentation[cardValue] = uRepresentation[cardValue].add
           (addition);
    }

    BigInteger getVectorPosition(int position) {
        if ( (position < 1) || (position > totalNumOfCards) ) { //
           bound check
            return null;
        }
        return uRepresentation[position];
    }

    CardVectorRepresentation(String inputString, int
       totalNumOfCards) throws Exception {
        String[] vectorItems = inputString.split(" ");
        if (vectorItems.length != totalNumOfCards) {
            throw new Exception("Illegal number of cards provided")
               ;
        }
        uRepresentation = new BigInteger[totalNumOfCards + 1];
        uRepresentation[0] = null;
        for (int i = 1; i <= totalNumOfCards; i++) {
            uRepresentation[i] = new BigInteger(vectorItems[i-1]);
        }
    }

    /*
     * Returns the product of card x Matrix
     * http://mathworld.wolfram.com/MatrixMultiplication.html
     */
```

```
    CardVectorRepresentation(CardVectorRepresentation card,
       CardPermutationMatrix matrix, int totalNumOfCards) {
        BigInteger[][] permMatrix = matrix.getPermutationMatrix();
        uRepresentation = new BigInteger[totalNumOfCards + 1];
        uRepresentation[0] = null;
        for (int calculatedIndex = 1; calculatedIndex <=
           totalNumOfCards; calculatedIndex++) {
            BigInteger columnSum = BigInteger.ZERO;
            for (int i=1; i <= totalNumOfCards; i++) {
                columnSum = columnSum.add(permMatrix[
                   calculatedIndex][i]);
            }
            uRepresentation[calculatedIndex] = columnSum.multiply(
               card.getVectorPosition(calculatedIndex));
        }
    }

    int getCardValue(BigInteger zPrime) {
        for (int i=1; i <= totalNumOfCards; i++) {
            BigInteger moduloZi = uRepresentation[i].mod(zPrime);
            if (moduloZi.compareTo(BigInteger.ZERO) == 1) {
                return i;
            }
        }
        return -1;
    }

    public String toString() {
        String output = "";
        for (int i = 1; i <= totalNumOfCards; i++) {
            output += uRepresentation[i].toString();
            if (i != totalNumOfCards) {
                output += " ";
            }
        }
        return output;
    }
}
```

6.4.4 implementation_code/java_code/DNChain.java

```
import java.util.ArrayList;

public class DNChain {
    public ArrayList<DataChainLink> linkArray;
```

```
    final static int ChainLength = 100;

    DNChain() {
        linkArray = new ArrayList<DataChainLink>(ChainLength);
    }

    public void add(DataChainLink addDataChainLink) {
        linkArray.add(addDataChainLink);
        //System.out.println("ChainSize:" + linkArray.size());
    }

    public void addArray(DataChainLink[] addDataChainLinkArray) {
        for (int i=0; i < addDataChainLinkArray.length; i ++) {
            add(addDataChainLinkArray[i]);
        }
    }

    public DataChainLink getLastLcaseBigoCard() {
        for (int i = (linkArray.size() - 1); i >= 0; i--) {
            DataChainLink currentCard = linkArray.get(i);
            if (currentCard.isLcaseBigOZero()) { //w_0
                return currentCard;
            }
        }
        return null; //nothing found

    }

    public int getSize() {
        return linkArray.size();
    }

    public String dumpChain() {
        String result = "";
        for (int i=0; i < linkArray.size(); i++) {
            DataChainLink currentLink = (DataChainLink) linkArray.
               get(i);
            result += "i: " + currentLink.forOutput() + "\n";
        }
        return result;
    }

    public int lastPlayerRequestedCard() {
        // TODO Auto-generated method stub
        return 0;
    }
```

```
    public DataChainLink getLastLinkOfPlayer(int playerLink) {
        for (int i = linkArray.size() - 1; i >= 0 ; i--) {
            DataChainLink currentLink = (DataChainLink) linkArray.
               get(i);
            if (currentLink.PlayerPosition == playerLink) {
                return currentLink;
            }
        }
        return null;
    }

    public boolean reveivedWZero() {
        DataChainLink lastLink = (DataChainLink) linkArray.get(
           linkArray.size() - 1);
        return (lastLink.isWZero());
    }

    public boolean receivedW_j_primePrevious(int position) {
        int playerToCheckChain = position - 1;
        if (playerToCheckChain < 1) {
            return false;
        }
        DataChainLink lastLink = (DataChainLink) linkArray.get(
           linkArray.size() - 1);
        if (lastLink.PlayerPosition == playerToCheckChain
            && lastLink.isWjprimeofPreviousPlayer(position) ) {
            return true;
        }
        return false;
    }

    public boolean receivedW_j(int position) {
        int playerToCheckChain = position - 1;
        if (playerToCheckChain < 1) {
            return false;
        }
        DataChainLink lastLink = (DataChainLink) linkArray.get(
           linkArray.size() - 1);
        if (lastLink.PlayerPosition == playerToCheckChain
            && lastLink.isWjofPreviousPlayer(position) ) {
            return true;
        }
        return false;
    }

//  public static void main(String args[]) {
//      DNChain dChain = new DNChain();
```

```
//
//      dChain.add(new DataChainLink(BigInteger.TEN, 999));
//
//      BigInteger sillyPrime = new BigInteger("7"); //this should
   originate from player's key generation
//      Deck initialPlayerDeck = new Deck();
//      CardPermutationMatrix initialPermMatrix = new
   CardPermutationMatrix(initialPlayerDeck, sillyPrime);
//      String commitment_P_i = initialPermMatrix.getCommitment();
//      int possition_P_i = 1;
//      String previousSignature = null;
//      System.out.println( commitment_P_i );
//
//      dChain.add(new DataChainLink(commitment_P_i,
   previousSignature, possition_P_i));
//  }
}
```

6.4.5 implementation_code/java_code/DataChainLink.java

```
import java.math.BigInteger;
import java.security.NoSuchAlgorithmException;

public class DataChainLink {

    public String ChainingValue;  // X_k
    public String TimeStamp;      // T_k
    public String Concept;        // C_k
    public String Attributes;     // V_k
    public int    PlayerPosition; //player's position in game

    private static String Seperator                 = " . ";
    private static String Concept_Prime             = "PZ_i";
    private static String Concept_Commitment        = "CP_i";
    private static String Concept_UcaseDeltaSet     = "BD_i";
    private static String Concept_UcaseEpsilonSet   = "UE_i";
    private static String Concept_EncryptedDeck     = "ED_i";
    private static String Concept_ChainContract     = "CC_i";
    private static String Concept_DummyLink         = "DM_i";
    private static String Concept_w0_Card_Draw      = "W0_i";
    private static String Concept_wi_Card_Draw      = "Wi_i";
    private static String Concept_wiPrime_Card_Draw = "Pi_i";

    private static int ChainElementsSize = 5;
```

```
    private static String EmptyChainingValue     = "nochainval";
    private static String ConcatenationSeperator = "-";

     /* Constructors */

    DataChainLink() { //returns "dummy" link with no significant
       properties
        this.PlayerPosition = -1;
        this.TimeStamp      = MPGame.getDateTime();
        this.Concept        = Concept_DummyLink;
        this.Attributes     = new String(EmptyChainingValue);
        this.ChainingValue  = new String(EmptyChainingValue); //
           signature of previous chainLink
    }

    DataChainLink(EncryptedCard lcaseBigOPrime, String
       prevSignature, int playerPos) {
        this.PlayerPosition = playerPos;
        this.TimeStamp      = MPGame.getDateTime();
        this.Concept        = Concept_wiPrime_Card_Draw;
        this.Attributes     = lcaseBigOPrime.forOutput();
        this.ChainingValue  = prevSignature; //signature of
           previous chainLink
    }

    DataChainLink (EncryptedVectorDeck encDeck, String
       prevSignature, int playerPos) {
        this.PlayerPosition = playerPos;
        this.TimeStamp      = MPGame.getDateTime();
        this.Concept        = Concept_EncryptedDeck;
        this.Attributes     = encDeck.encOutput();
        this.ChainingValue  = prevSignature; //signature of
           previous chainLink
    }


    DataChainLink (DeltaEpsilonSet deSet, boolean isDelta, String
       prevSignature, int playerPos) {
        this.PlayerPosition = playerPos;
        this.TimeStamp = MPGame.getDateTime();
        this.Concept   = (isDelta)? new String(
           Concept_UcaseDeltaSet) : new String(
           Concept_UcaseEpsilonSet);
        this.Attributes = deSet.output();
        this.ChainingValue = prevSignature; //signature of previous
            chainLink
    }
```

```
    DataChainLink (BigInteger Z_i, int playerPos) { //For initial
       chain link, builds it based on prime Z of player i
        this.PlayerPosition = playerPos;
        this.TimeStamp = MPGame.getDateTime();
        this.Concept   = new String(Concept_Prime);
        this.Attributes = Z_i.toString();
        this.ChainingValue = EmptyChainingValue; //first link has
           no chaining value
    }

    DataChainLink (String playerCommitment, String prevSignature,
       int playerPos) {
        this.PlayerPosition = playerPos;
        this.TimeStamp = MPGame.getDateTime();
        this.Concept   = new String(Concept_Commitment);
        this.Attributes = playerCommitment;
        this.ChainingValue = prevSignature; //signature of previous
            chainLink
    }

    DataChainLink(DataChainLink[] contractionLinks, MPElGamal
       cryptoSystem, int playerPos) { //note: no signature
        this.PlayerPosition = playerPos;
        this.TimeStamp = MPGame.getDateTime();
        this.Concept = new String(Concept_ChainContract);
        this.Attributes = "";
        for (int i = 1; i < contractionLinks.length; i++) {
            this.Attributes += contractionLinks[i].Attributes;
            if (i != (contractionLinks.length - 1)) {
                this.Attributes += ConcatenationSeperator;
            }
        }

        // compute hash
        byte[] attributesHash = null;
        try {
            attributesHash = Signature.getSHA1Byte(this.Attributes)
               ;
        } catch (NoSuchAlgorithmException algExp) {
            System.err.println("Sha1 not supported");
            System.exit(-2);
        }
        MPSignedInteger signedHash = cryptoSystem.sign(
           attributesHash);
        String signedHashStr = signedHash.strOutput();
        ; // sign hash
```

```
        this.ChainingValue = signedHashStr;
    }

    //DataChain Link from u0/uj integer:
    DataChainLink (CardVectorRepresentation cardVect, boolean
       w0Flag,
            String prevSignature, int playerPos) {
        this.PlayerPosition = playerPos;
        this.TimeStamp = MPGame.getDateTime();
        if (w0Flag == true) {
            this.Concept = new String(Concept_w0_Card_Draw);
        } else {
            this.Concept = new String(Concept_wi_Card_Draw);
        }
        this.Attributes = cardVect.toString();
        this.ChainingValue = prevSignature; //signature of previous
            chainLink
    }

    public static DataChainLink fromString(String strInput) {
        System.out.println("input: " + strInput);
        DataChainLink dnc = new DataChainLink();
        String dncElements[] = strInput.split(Seperator);
        if (dncElements.length != ChainElementsSize) { //error
            System.out.println("Illegal number of elements submited
                in chain link ( "
                    + dncElements.length + ").");
            System.out.println("Firs element: " + dncElements[0]);

            return null;
        }
        try {
            dnc.PlayerPosition = Integer.parseInt(dncElements[0]);
            dnc.TimeStamp = dncElements[1];
            dnc.Concept = dncElements[2];
            dnc.Attributes = dncElements[3];
            dnc.ChainingValue = dncElements[4];
        } catch (Exception exp) {
            System.out.println("Exception in parsing.");
            return null;
        }
        return dnc;
    }


    /**
     * For communication among Players
```

```
     * @return comma separated  output of DataChainLink for
        transmission
     */
    public String forOutput() {
        String output = PlayerPosition + Seperator
            + TimeStamp + Seperator
            + Concept + Seperator
            + Attributes + Seperator
            + ChainingValue;
        return output;
    }

    public String getSeperator() {
        return Seperator;
    }

    public boolean isDummy() {
        return (this.Concept == Concept_DummyLink);
    }

    public boolean isLcaseBigOZero() {
        return (this.Concept == Concept_w0_Card_Draw);
    }

    public boolean isLcaseBigOZeroPrime() {
        return (this.Concept == Concept_wiPrime_Card_Draw);
    }

    public boolean isWjofPreviousPlayer(int position) {
        return (this.Concept == Concept_wi_Card_Draw) && (this.
           PlayerPosition == (position -1));
    }

    public boolean isWjprimeofPreviousPlayer(int position) {
        return (this.Concept == Concept_wiPrime_Card_Draw) && (this
           .PlayerPosition == (position -1));
    }

    public boolean isWZero() {
        return (this.Concept == Concept_w0_Card_Draw);
    }

    public CardVectorRepresentation getCardVectorRepresentation() {
        if (this.Concept != this.Concept_wi_Card_Draw) {
            return null;
        }
        // TODO Auto-generated method stub
```

```
        return null;
    }
}
```

6.4.6 implementation_code/java_code/Deck.java

```
public class Deck {
    final public static int numberOfCards = 5;

    private int[] initialPermutation;
    /**
     * Generates a permutation of the card deck
     */
    Deck() {
        initialPermutation = new int[numberOfCards + 1];
        for (int i = 1; i <= numberOfCards; i++) { // create
            initialPermutation[i] = i;
        }
        this.shuffle();
    }

    public Deck shuffle(Deck inputDeck) {
        Deck newDeck = new Deck();
        for (int i = 0; i < numberOfCards; i++) { // copy
            newDeck.initialPermutation[i] = inputDeck.
               initialPermutation[i];
        }
        newDeck.shuffle();
        return newDeck;
    }


    public int getCardAtPosition(int i) {
        if ( (i<=numberOfCards) && (i>0) ) {
            return initialPermutation[i];
        } else {
            return -1;
        }
    }

    /* Code from this originated from: http://www.cs.princeton.edu/
       introcs/21function/Shuffle.java.html
     * Version here has been extensively rewritten and does not
        look like the referenced one.
     */
```

```
    protected void shuffle() {
        for (int i = 1; i <= numberOfCards; i++) {
            int r = i + (int) (Math.random() * (numberOfCards-i));
                 // between i and N-1
            exch(i, r);
        }
        //Debug Code:
        //for (int i=0; i < numberOfCards; i++) { System.out.
           println( i + "card: " + initialPermutation[i]); }
    }

    // swaps array elements i and j
    protected void exch(int i, int j) {
        int swap = initialPermutation[i];
        initialPermutation[i] = initialPermutation[j];
        initialPermutation[j] = swap;
    }
}
```

6.4.7 implementation_code/java_code/DeltaEpsilonSet.java

```
import java.math.BigInteger;

class DeltaEpsilonSet {
    final static String Seperator = ", ";
    final static String InternalSeperator = "-";

    MPEncryptedMessage[] BigDelta;

    DeltaEpsilonSet(LcaseDeltaSet lDelta, MPElGamal cryptoSystem) {
        BigDelta = new MPEncryptedMessage[LcaseDeltaSet.SizeS];
        BigInteger msgToEncrypt;
        for (int i=0; i < LcaseDeltaSet.SizeS; i++) {
            //System.out.println("position: " + i);
            msgToEncrypt = lDelta.getItem(i);
            //System.out.println("  got item: " + msgToEncrypt);
            BigDelta[i] = cryptoSystem.encrypt(msgToEncrypt);
        }
    }

    public String output() {
        String result = "";
        for (int i=0; i < LcaseDeltaSet.SizeS; i++) {
            result += BigDelta[i].strOutput();
            if ( i != (LcaseDeltaSet.SizeS - 1) ) {
```

```
                result += Seperator;
            }
        }
        return result;
    }

    public int getSize() {
        return BigDelta.length;
    }

    public MPEncryptedMessage getItem(int position) {
        if ( (position < 0) || (position >= BigDelta.length) ) {
            return null;
        }
        return BigDelta[position];
    }


}
```

6.4.8 implementation_code/java_code/EcnryptedDeck.java

```
class EcnryptedDeck {
    public EncryptedCard[] encCards;

    EcnryptedDeck() {
        encCards = new EncryptedCard[Deck.numberOfCards];
    }

    public String forOutput() {
        String result = "";
        for (int i=0; i < Deck.numberOfCards; i++) {
            result += encCards[i].forOutput();
            if (i != (Deck.numberOfCards - 1)) {
                result += EncryptedCard.Seperator; //using the same
                    ,
                //since the other part knows it's length and can
                    therefore decode
            }
        }
        return result;
    }
}
```

6.4.9 implementation_code/java_code/ElGamal.java

```
// http://faculty.washington.edu/moishe/javademos/Security/ElGamal.
   java

import java.math.*;
import java.util.*;
import java.security.*;
import java.io.*;

public class ElGamal
{
    public static void main(String[] args) throws IOException
    {
        BigInteger p, b, c, secretKey;
        Random sc = new SecureRandom();
        secretKey = new BigInteger("12345678901234567890");
        //
        // public key calculation
        //
        System.out.println("secretKey = " + secretKey);
        p = BigInteger.probablePrime(64, sc);
        b = new BigInteger("3");
        c = b.modPow(secretKey, p);
        System.out.println("p = " + p);
        System.out.println("b = " + b);
        System.out.println("c = " + c);
        //
        // Encryption
        //
        System.out.print("Enter your Big Number message -->");
        String s = Tools.GetString();
        BigInteger X = new BigInteger(s);
        BigInteger r = new BigInteger(64, sc);
        BigInteger EC = X.multiply(c.modPow(r, p)).mod(p);
        BigInteger brmodp = b.modPow(r, p);
        System.out.println("Plaintext = " + X);
        System.out.println("r = " + r);
        System.out.println("EC = " + EC);
        System.out.println("b^r mod p = " + brmodp);
        //
        // Decryption
        //
        BigInteger crmodp = brmodp.modPow(secretKey, p);
        BigInteger d = crmodp.modInverse(p);
```

```
        BigInteger ad = d.multiply(EC).mod(p);
        System.out.println("\n\nc^r mod p = " + crmodp);
        System.out.println("d = " + d);
        System.out.println("Alice decodes: " + ad);

    }
}
```

6.4.10 implementation_code/java_code/EncryptedCard.java

```
import java.math.BigInteger;

class EncryptedCard {
    public final static String Seperator = ", ";
    MPEncryptedMessage[] encryptedVector;

    public EncryptedCard(CardVectorRepresentation inputCard,
       MPElGamal cryptoSystem) {
        encryptedVector = new MPEncryptedMessage[Deck.numberOfCards
            + 1];
        encryptedVector[0] = null;
        for (int i=1; i <= Deck.numberOfCards; i++) {
            BigInteger currentBI = inputCard.getVectorPosition(i);
            encryptedVector[i] = cryptoSystem.encrypt(currentBI);
        }
    }

    public EncryptedCard(EncryptedCard muptiplyCard,
       EncryptedPermutationMatrix encMatrix) {
        MPEncryptedMessage[] cardVector = muptiplyCard.
           getVectorArray();
        int totalNumOfCards = cardVector.length;
        MPEncryptedMessage[][] encMatrixArray =encMatrix.
           getMatrixArray();
        encryptedVector = new MPEncryptedMessage[totalNumOfCards +
           1];
        encryptedVector[0] = null;

        for (int calculatedIndex = 1; calculatedIndex <=
           totalNumOfCards; calculatedIndex++) {
            MPEncryptedMessage columnSum = new MPEncryptedMessage(
               BigInteger.ZERO, BigInteger.ZERO);
            for (int i=1; i <= totalNumOfCards; i++) {
                BigInteger a = encMatrixArray[calculatedIndex][i].
                   encA;
```

```
                BigInteger b = encMatrixArray[calculatedIndex][i].
                    encB;
                columnSum.add(a, b);
            }
            encryptedVector[calculatedIndex] = new
                MPEncryptedMessage(cardVector[calculatedIndex].encA,
                cardVector[calculatedIndex].encB);
            encryptedVector[calculatedIndex].multiply(columnSum);
        }
    }

    public MPEncryptedMessage[] getVectorArray() {
        return this.encryptedVector;
    }

    public String forOutput() {
        String result = "";
        for (int i=1; i <= Deck.numberOfCards; i++) {
            result += encryptedVector[i].strOutput();
            if (i != (Deck.numberOfCards - 1)) {
                result += Seperator;
            }
        }
        return result;
    }
}
```

6.4.11 implementation_code/java_code/EncryptedPermutationMatrix.java

```
import java.math.BigInteger;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;

public class EncryptedPermutationMatrix {
    protected MPEncryptedMessage[][] encPermMatrix;
    SecureRandom sec;

    EncryptedPermutationMatrix(
            CardPermutationMatrix permMatrix,
            DeltaEpsilonSet deltaSet,
            DeltaEpsilonSet epsilonSet,
            MPElGamal crypto,
            SecureRandom sec,
            BigInteger primeZ,
            BigInteger otherPrimeZ,
```

```
            int numberOfCards) {
        if (sec == null) {
            sec = new SecureRandom();
        }
        encPermMatrix = new MPEncryptedMessage[numberOfCards][
           numberOfCards];
        BigInteger[][] biPermMatrix = permMatrix.
           getPermutationMatrix();
        this.sec = sec;
        int s = deltaSet.getSize();
        for (int kLoop = 1; kLoop <= numberOfCards; kLoop++) {
            for (int lLoop = 1; lLoop <= numberOfCards; lLoop++) {
                mainEcnLoop(kLoop, lLoop, biPermMatrix, deltaSet,
                   epsilonSet, sec, primeZ, otherPrimeZ, s);
            }
        }
    }

    public MPEncryptedMessage[][] getMatrixArray() {
        return this.encPermMatrix;
    }


    private void mainEcnLoop(int kLoop, int lLoop, BigInteger[][]
       biPermMatrix,
            DeltaEpsilonSet deltaSet,
            DeltaEpsilonSet epsilonSet,
            SecureRandom sec,
            BigInteger primeZ, BigInteger otherPrimeZ, int s) {
        //generate pseudorandom value g
        int g = generatePseudoG(sec, s);
        //get g random numbers from 1 to s
        int deltaSetRandomPositions[] = new int[g];
        for (int i=0; i < g; i++) {
            boolean existsInArray = false;
            do {
                existsInArray = false;
                int nextInt = sec.nextInt(s);
                for (int j=0; j < i; j++) {
                    if (nextInt == deltaSetRandomPositions[j]) {
                        existsInArray = true;
                    }
                }
                deltaSetRandomPositions[i] = nextInt;
            } while (!existsInArray);
        }
```

```
        MPEncryptedMessage h = new MPEncryptedMessage(BigInteger.
            ZERO, BigInteger.ZERO);
        for (int i=0; i < g; i++) {
            MPEncryptedMessage summationElement = deltaSet.getItem(
                deltaSetRandomPositions[i]);
            h.add(summationElement);
        }
        BigInteger c = get_c(primeZ);
        MPEncryptedMessage hPrime = new MPEncryptedMessage(h.encA,
            h.encB);
        hPrime.mutiply(c);

        //steps 4 and 5:
        if (biPermMatrix[kLoop][lLoop].mod(otherPrimeZ).equals(
            BigInteger.ZERO)) { //step 4
            encPermMatrix[kLoop][lLoop] = hPrime;
        } else { //step 5
            int gPrime = this.generatePseudoG(sec, s);
            MPEncryptedMessage gThElementofEpsilon = epsilonSet.
                getItem(gPrime);
            MPEncryptedMessage hPrimePlusGth = new
                MPEncryptedMessage(hPrime.encA, hPrime.encB);
            hPrimePlusGth.add(gThElementofEpsilon);
            encPermMatrix[kLoop][lLoop] = hPrimePlusGth;
        }
    }


    private int generatePseudoG(SecureRandom sec, int s) {
        int g = sec.nextInt();
        if (g > s) { //trim it
            g = g % s;
        }
        if (g == 0) {
            g = 1;
        }
        return g;
    }


    private BigInteger get_c(BigInteger z) {
        //return random c: c mod z != 0
        int nextInt = sec.nextInt(100); //suppose between one and
            100
        if (nextInt == 0) {
            nextInt = 1;
        }
```

```
        BigInteger nextBI = BigInteger.valueOf((long)nextInt);
        return nextBI.multiply(z);
    }

}
```

6.4.12 implementation_code/java_code/EncryptedVectorDeck.java

```
import java.math.BigInteger;
/*
 * Represents a deck of cards under the Card Vector Representation
 */
class EncryptedVectorDeck {
    private EncryptedCard[] encCard;

    public EcnryptedDeck encryptedVDeck;

    final static String Seperator = ", ";

    public EncryptedVectorDeck(VectorDeck inputVDeck, BigInteger zI
        , MPElGamal myCrypto) {

        encCard = new EncryptedCard[Deck.numberOfCards + 1];
        encCard[0] = null;
        for (int i=1; i <= Deck.numberOfCards; i++) {
            encCard[i] = new EncryptedCard(inputVDeck.
                getCardAtPosition(i), myCrypto);
        }

//      vDeck = new Card[Deck.numberOfCards];
//      for (int i=0; i < Deck.numberOfCards; i++) {
//          vDeck[i] = new Card(inputDeck.getCardAtPosition(i), zI,
    Deck.numberOfCards);
//      }
//      encryptedVDeck = null;
    }

//  public void encrypt(MPElGamal cryptoSystem) {
//      if (encryptedVDeck == null) {
//          encryptedVDeck = new EcnryptedDeck();
//          for (int i=0; i < Deck.numberOfCards; i++) {
//              encryptedVDeck.encCards[i] = new EncryptedCard(
   vDeck[i], cryptoSystem);
//          }
//      }
```

```
//  }

    public String encOutput() {
        String result = "";
        for (int i =1; i <= Deck.numberOfCards; i++) {
            result += encCard[i].forOutput();
            if (i != Deck.numberOfCards) {
                result += Seperator;
            }
        }
        return result;
    }

    public void shuffle() {
        for (int i = 1; i <= Deck.numberOfCards; i++) {
            int r = i + (int) (Math.random() * (Deck.numberOfCards-
               i));   // between i and N-1
            exch(i, r);
        }
//         //Debug Code
//        for (int i=0; i < Deck.numberOfCards; i++) {
//            System.out.println( i + "card: " + encryptedVDeck.
   encCards[i].forOutput());
//        }
    }

    // swaps array elements i and j
    protected void exch(int i, int j) {
        EncryptedCard swap = this.encCard[i];
        this.encCard[i] = this.encCard[j];
        this.encCard[j] = swap;
    }

    public EncryptedCard getCardAtPosition(int position) {
        if ( (position < 1) || (position > Deck.numberOfCards) ) {
            return null;
        }
        return encCard[position];
    }

}
```

6.4.13 implementation_code/java_code/InitializationState.java

```
```

```
public enum InitializationState {
    START,
    ZIBROADCAST,
    PERMUTATIONMATRIXBCAST,
    UCASEDELTASETBCAST,
    BIGEPSILONSETBCAST,
    VECTORDECKBCAST,
    PLAYERCROUPIERBCAST,
    END
}
```

6.4.14 implementation_code/java_code/LcaseDeltaSet.java

```
import java.math.BigInteger;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;


/**
 * Used for h part of initialization algorithm computes a set of
    small deltas
 * @author dimitris
 *
 */
class LcaseDeltaSet {
    final static int SizeS = Deck.numberOfCards + 2; //s such as s
       > t (t==numberOfCards)
    final static int RandomNumberOfBits = 5;

    protected BigInteger[] numCollection;

    /**
     *
     * @param primeZi prime number Z chosen by current player
     */
    LcaseDeltaSet(BigInteger primeZ) {
        numCollection = new BigInteger[SizeS];
        SecureRandom secRnd = new SecureRandom();
        for (int i=0; i < SizeS; i++) {
            BigInteger randomBI = new BigInteger(RandomNumberOfBits
               , secRnd);
            randomBI = randomBI.abs();
            numCollection[i] = randomBI.multiply(primeZ);
        }
    }
```

```
    BigInteger getItem(int position) {
        if ( (position < 0) || (position >= SizeS) ) {
            return null;
        }
        return numCollection[position];
    }

}
```

6.4.15 implementation_code/java_code/LcaseEpsilonSet.java

```
import java.math.BigInteger;
import java.security.SecureRandom;

class LcaseEpsilonSet extends LcaseDeltaSet {

    LcaseEpsilonSet(BigInteger primeZ) {
        super(primeZ);
        SecureRandom secRnd = new SecureRandom();
        for (int j=0; j < SizeS; j++) {
            BigInteger randomBI = new BigInteger(RandomNumberOfBits
                , secRnd);
            randomBI = randomBI.abs();
            randomBI = randomBI.mod(primeZ);
            numCollection[j] = numCollection[j].add(randomBI);
        }
    }

}
```

6.4.16 implementation_code/java_code/MPElGamal.java

```
import java.math.BigInteger;
import java.security.*;
import java.security.interfaces.*;

/*
 * Custom implementation of ElGamal
 * Ideas, but not copy-paste from:
 * http://faculty.washington.edu/moishe/javademos/Security/ElGamal.
    java
 *
```

```
 * Proof of homomorphic property of ElGamal's encryption
 * http://www.cs.ucla.edu/~rafail/TEACHING/WINTER-2005/L8/L8.ps
 * TODO: check this as well: http://developer.berlios.de/projects/
    elgamal/
 */
public class MPElGamal {
    private final int RandomBitLength = 50; //randomly chosen
    private final int CryptographyError = -3;

    private MPKeyPrivate privateKey;
    private MPKeyPublic  publicKey;
    private BigInteger zPlayer = null;

    public static int Apos = 0;
    public static int Bpos = 1;
    public static int EncryptionArraySize = 2;
    public static final String Seperator = "-";

    //missing:
    //signature verification

    public MPKeyPublic getPublicKey() {
        return publicKey;
    }

    public MPKeyPrivate getPrivateKey() {
        return privateKey;
    }

    public BigInteger getZPlayer() {
        if (zPlayer == null) {
            return generateZ();
        } else {
            return zPlayer;
        }
    }

    private BigInteger generateZ() {
        zPlayer = BigInteger.ZERO;
        int zLegth = publicKey.p.bitLength();
        SecureRandom sec = new SecureRandom();
        do {
            zPlayer = new BigInteger(zLegth, sec);
        } while ((zPlayer.compareTo(publicKey.p) >= 0) || (zPlayer.
           compareTo(BigInteger.ZERO) <= 0));
        return zPlayer;
    }
```

```
    MPElGamal() {
        //initialization
        SecureRandom sec = new SecureRandom();
        publicKey  = new MPKeyPublic();
        privateKey = new MPKeyPrivate();
        //assignments
        publicKey.p  = new BigInteger(RandomBitLength, sec);
        publicKey.g  = new BigInteger(RandomBitLength - 2 , sec);
        privateKey.x = new BigInteger(RandomBitLength - 2 , sec);
        publicKey.y  = publicKey.g.modPow(privateKey.x, publicKey.p
           );
    }

    public MPEncryptedMessage encrypt(BigInteger message) {
        BigInteger pMinusOne = (publicKey.p).add(BigInteger.ONE.
           negate());
        BigInteger randomK;
        randomK = randomRelativelyPrime(pMinusOne);
        BigInteger resultA;
        BigInteger resultB;

        resultA =  (publicKey.g).modPow(randomK, publicKey.p);
        /*
         * b = ( (y^k)*M ) % p =
         *   = ( ((y^k) % p) * (M % p) ) % p
         *       ^^^^^= ypowk  ^^^^=mpowk
         */
        BigInteger ypowk = (publicKey.y).modPow(randomK, publicKey.
           p);
        BigInteger mpowk = message.mod(publicKey.p);
        resultB = ( ypowk.multiply(mpowk) ).mod(publicKey.p);
        MPEncryptedMessage mpEncRes = new MPEncryptedMessage(
           resultA, resultB);
        return mpEncRes;
    }


    public BigInteger decrypt(MPEncryptedMessage mpEnc) {
        //message = (b /a ^ x) mod p =
        //        = (bmodp) / ( (a^x mod p) ) mod p

        BigInteger aPowX = (mpEnc.encA).modPow(privateKey.x,
           publicKey.p);
        BigInteger aPowXInv = aPowX.modInverse(publicKey.p);
        BigInteger bModp = (mpEnc.encB).mod(publicKey.p);
        BigInteger division = bModp.multiply(aPowXInv);
```

```
        BigInteger message = division.mod(publicKey.p);
        return message;
    }

    public String signString(String message) {
        //getSHA1Byte
        byte[] hashedMessageByte = null;
        try {
            hashedMessageByte = Sha1Signature.getSHA1Byte(message);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            System.err.println("No Sha1 supported, exiting");
            System.exit(CryptographyError);
        }
        MPSignedInteger signedMessage = sign(hashedMessageByte);
        return signedMessage.strOutput();
    }

    public MPSignedInteger sign(byte[] hashedMessageByte) {

        BigInteger hashedMessageBI = new BigInteger(
           hashedMessageByte);

        BigInteger pMinusOne = (publicKey.p);
        pMinusOne = pMinusOne.subtract(BigInteger.ONE);

        if (hashedMessageBI.compareTo(pMinusOne) == 1) {
            System.err.print("messageBI: " + hashedMessageBI
                    + " is bigger than pMinusOne, " + pMinusOne + "
                       , proceeding with mod (" );
            hashedMessageBI = hashedMessageBI.mod(pMinusOne);
            System.err.println( hashedMessageBI + ")");
        }

        BigInteger randomK = randomRelativelyPrime(pMinusOne);
        BigInteger kPowMinOne = randomK.modInverse(pMinusOne);

        BigInteger a = (publicKey.g).modPow(randomK, (publicKey.p))
           ;

        //steps for calculating b:
        BigInteger amulr = a.multiply(randomK);
        BigInteger hashMinusAmulR = hashedMessageBI.min(amulr);
        BigInteger b = kPowMinOne.multiply(hashMinusAmulR).mod(
           pMinusOne);
```

```
        MPSignedInteger signedMessage = new MPSignedInteger(a, b);
        return signedMessage;
    }

    private BigInteger randomRelativelyPrime(BigInteger coPrime) {
       //coPrime is modulo as well
        SecureRandom sec = new SecureRandom();
        BigInteger gcdRandomCoPrime = null;
        BigInteger randomK = null;
        do {
            randomK = new BigInteger(RandomBitLength, sec);
            randomK = randomK.abs();
            randomK = randomK.mod(coPrime);
            gcdRandomCoPrime = randomK.gcd(coPrime);
            //System.out.println("random: " + randomK + ",
               comparison result: " + gcdRandomCoPrime.compareTo(
               BigInteger.ONE));
        } while ( gcdRandomCoPrime.compareTo(BigInteger.ONE) != 0);
        return randomK;
    }

    /**
     * Generates an Elgamal private and public key and displays it
     * @param args
     */
    public static void main(String args[]) {
        MPElGamal mpCrypto = new MPElGamal();
        System.out.println("zPlayer: " + mpCrypto.getZPlayer());
        // try to encrypt and decrypt a big integer:
        BigInteger original = new BigInteger("42");
        BigInteger compare  = BigInteger.ZERO;

        MPEncryptedMessage mpEnc = mpCrypto.encrypt(original);
        compare = mpCrypto.decrypt(mpEnc);
        //encrypt
        //decrypt
        if (original.equals(compare)) {
            System.out.println("El Gamal test provided correct
               results");
        } else {
            System.err.println("El Gamal test did not provide
               correct results");
            System.err.println("original: " + original);
            System.err.println("compare : " + compare);
        }
```

```
    }
}

/*
 * Data Structures for keys
 */
class MPKeyPrivate {
    public BigInteger x; //El Gamal's x, random number less than p
}

class MPKeyPublic {
    public BigInteger p; //El Gamal's p, prime number
    public BigInteger g; //El Gamal's g, random number less than p
    public BigInteger y; //El Gamal's y, y = (g^x)mod(p)
}

class MPSignedInteger extends MPEncryptedMessage {
    MPSignedInteger(BigInteger a, BigInteger b) {
        super(a, b);
    }
}
```

6.4.17 implementation_code/java_code/MPEncryptedMessage.java

```
import java.math.BigInteger;

class MPEncryptedMessage {
    public BigInteger encA; //ElGamal's A
    public BigInteger encB; //ElGamal's B

    MPEncryptedMessage(BigInteger a, BigInteger b) {
        encA = a;
        encB = b;
    }

    public String strOutput() {
        return encA + MPElGamal.Seperator + encB;
    }

    public void add(MPEncryptedMessage summationElement) {
        encA = encA.add(summationElement.encA);
        encB = encB.add(summationElement.encB);
    }
```

```
    public void mutiply(BigInteger c) {
        encA = encA.multiply(c);
        encB = encB.multiply(c);
    }

    public void add(BigInteger a, BigInteger b) {
        encA = encA.add(a);
        encB = encB.add(b);
    }

    public void multiply(MPEncryptedMessage mpEncryptedMessage) {
        encA = encA.multiply(mpEncryptedMessage.encA);
        encB = encB.multiply(mpEncryptedMessage.encB);
    }
}
```

6.4.18 implementation_code/java_code/MPGame.java

```
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;


public class MPGame {

    final static int numberOfPlayers = 5;

    /**
     * @param args
     */
    public static void main(String[] args) {
        Player[] player = new Player[numberOfPlayers + 1]; //
           emulate array from 1 to numberOfPlayers
        player[0] = null; //we start from 1
        for (int i=1; i <= numberOfPlayers; i++) {
            player[i] = new Player(i);
        }

        boolean endOfGameFlag = false;
        do {
            //This is a no-thread imitation of broadcast
               communication,
            //which has to do with algorithm and JIF-specific
               issues
            int nullLinkCount = 0;
```

```
            for (int i=1; i <= numberOfPlayers; i++) {
                DataChainLink broadcastedDNCLink = player[i].
                    getNextDNCLink();
                if (broadcastedDNCLink != null) {
                    for (int j=1; j <= numberOfPlayers; j++) {
                        player[j].receiveNextDNCLink(
                            broadcastedDNCLink);
                    }
                } else {
                    nullLinkCount++;
                    if (nullLinkCount == numberOfPlayers) { //
                        nothing new from ALL players
                        endOfGameFlag = true;
                    }
                }
            }

        } while (!endOfGameFlag);

        //see status:
        for (int i=1; i <= numberOfPlayers; i++) {
            System.out.println( player[i].dump() );
        }
    }


    //Common utility functions:

    public static String getDateTime() {
        DateFormat dateFormat = new SimpleDateFormat("
            yyyyMMddHHmmss");
        Date date = new Date();
        return dateFormat.format(date);
    }
}
```

6.4.19 implementation_code/java_code/Player.java

```
import java.math.BigInteger;

public class Player {
    private int position = -1;
    private String name = null;
    private Deck mySecretDeck = null;
    private BigInteger primeNumberZ = null;
```

```
    private MPElGamal myCrypto = null;
    private CardPermutationMatrix myPermMatrix = null;
    private InitializationState initializationState =
       InitializationState.START;
    private CardDrawState cardDrawState = CardDrawState.PLAYERIDLE;
    private String lastSignature = null;
    private DNChain myChain = null;
    private DNChain otherPlayersChain = null;
    private LcaseDeltaSet myLcaseDeltaStet = null;
    private LcaseEpsilonSet myLcaseEpsilonStet = null;
    private DeltaEpsilonSet myUcaseDeltaSet = null;
    private DeltaEpsilonSet myUcaseEpsilonSet = null;
    private VectorDeck myVectorDeck = null;
    private EncryptedVectorDeck myEncryptedVectorDeck = null;
    private String dncConcatenation;

    //Card Draw phase:
    final static int NumberOfCardsToDraw = 3; //1st assumption on
       running the algorithm
    final static int CardDrawerPosition  = 3; //2nd assumption on
       running the algorithm
    private int cardsDrawn = 0;
    private boolean haveBroadcastesBigOZeroPrime = false;
    private boolean cardDrawBroadCastFlag = false;

    UZeroGenerator uZeroGen = null;

    public Player(int position) {
        this.position = position;
        this.name = position + "-player";

        System.out.println("Initialization of" + this.name);
        mySecretDeck      = new Deck(); //Generate Permutation
        myChain           = new DNChain();
        otherPlayersChain = new DNChain();
        //initialize Cryptographic System:
        myCrypto = new MPElGamal();
        primeNumberZ = myCrypto.getZPlayer();

        //calculate small_delta (\delta) and small_epsilon (\
           epsilon) sets

        myLcaseDeltaStet   = new LcaseDeltaSet(primeNumberZ); //(h)
            Choose s values... low Delta Set
        myLcaseEpsilonStet = new LcaseEpsilonSet(primeNumberZ); //(
           i) Choose s values... low Epsilon Set
```

```
        myUcaseDeltaSet = new DeltaEpsilonSet(myLcaseDeltaStet,
            myCrypto); //(j)Encrypting Small Delta
        myUcaseEpsilonSet = new DeltaEpsilonSet(myLcaseEpsilonStet,
            myCrypto); //(j)Encrypting Small Epsilon


        myVectorDeck = new VectorDeck(mySecretDeck, primeNumberZ);
            //(m.a) of algorithm: Generate the vector
               representation of cards of Deck

        //calculate permutation matrix:
        myPermMatrix = new CardPermutationMatrix(mySecretDeck,
            primeNumberZ);

        myEncryptedVectorDeck = new EncryptedVectorDeck(
            myVectorDeck, primeNumberZ, myCrypto);
            //(m.b) encrypt
        myEncryptedVectorDeck.shuffle(); //(n) permute

        uZeroGen = new UZeroGenerator(Deck.numberOfCards); //for
            card draw phase

    }

    public String dump() { //shows a summary of player's status
        String result = "Name: " + name + "\n";
        result += "myZ:                              " + primeNumberZ.
            toString()  + "\n";
        result += "my chain length:                  " + myChain.
            getSize()  + "\n";
        result += "other players chain length:    " +
            otherPlayersChain.getSize()  + "\n";
        result += "permutation matrix commitment: " + myPermMatrix.
            getCommitment()  + "\n";

        //result += "myChainDump:\n" + otherPlayersChain.dumpChain
            () + "\n";
        return result;
    }

    public DataChainLink getNextDNCLink() {
        DataChainLink nextLink;
        if ( intializationPhase() ) { //player in initialization
            phase.
            switch (initializationState) {
            case START:
                System.out.println("Start State");
```

```
                initializationState = InitializationState.
                    ZIBROADCAST; //next state
                nextLink = new DataChainLink(primeNumberZ, position
                    ); //generate Z_i DNC Link
                break;
            case ZIBROADCAST:
                System.out.println("Broadcasted Z_i");
                initializationState = InitializationState.
                    PERMUTATIONMATRIXBCAST;
                nextLink = new DataChainLink(myPermMatrix.
                    getCommitment(), lastSignature, position);
                break;
            case PERMUTATIONMATRIXBCAST:
                initializationState = InitializationState.
                    UCASEDELTASETBCAST;
                nextLink = new DataChainLink(myUcaseDeltaSet, true,
                     lastSignature, position);
                break;
            case UCASEDELTASETBCAST:
                initializationState = InitializationState.
                    BIGEPSILONSETBCAST;
                nextLink = new DataChainLink(myUcaseEpsilonSet,
                    false, lastSignature, position);
                break;
            case BIGEPSILONSETBCAST:
                initializationState = InitializationState.
                    VECTORDECKBCAST;
                //(o) Chain link of encrypted deck
                nextLink = new DataChainLink(myEncryptedVectorDeck,
                     lastSignature, position);
                break;
            case VECTORDECKBCAST:
                if (isCroupier()) {
                    initializationState = InitializationState.
                        PLAYERCROUPIERBCAST;
                    System.out.println("I am the Croupier and
                        buinding a concatenation link. This sigmals
                        the initialization phase");
                    nextLink = getInitializationContractionLink();
                } else {
                    initializationState = InitializationState.END;
                    return null;
                }
                break;
            case PLAYERCROUPIERBCAST:
                initializationState = InitializationState.END;
                return null;
```

```
            case END:
            default: //termination
                return null;
            }

        } else { //card draw
            //System.out.println("Card Draw phase");
            switch(cardDrawState) {
            case PLAYERIDLE:
                if (turnToDraw()) {
                    cardDrawState = CardDrawState.
                        REQUESTCARDWAITWMINUSONE;
                    nextLink = getWzeroDNC();
                    break;
                } else if ((position == 1) && this.received_w0()) {
                    //first player and received w0
                    nextLink = getWjLink();
                    cardDrawBroadCastFlag = true;
                    break;
                } if (this.received_w_j() && position <
                    otherPlayersChain.lastPlayerRequestedCard()) {
                    cardDrawState = CardDrawState.
                        PLAYERJRCVJMINUSONE;
                    nextLink = new DataChainLink(); //dummy
                    break;
                } if (this.received_w_j_prime() && position >
                    otherPlayersChain.lastPlayerRequestedCard()) {
                    cardDrawState = CardDrawState.
                        PLAYERJRCVWJPRIMEMINUSONE;
                    nextLink = new DataChainLink(); //dummy
                    break;
                } else {
                    return null;// nothing to do (store and stay
                        idle)
                }
            case PLAYER1RCVW0:
                nextLink = null; //send nothing
                cardDrawState = CardDrawState.PLAYERIDLE; //return
                    to IDLE state
                break;
            case PLAYERJRCVJMINUSONE:
                nextLink = null; //send nothing
                cardDrawState = CardDrawState.PLAYERIDLE; //return
                    to IDLE state
                break;
            case PLAYERJRCVWJPRIMEMINUSONE:
                nextLink = null; //send nothing
```

```
                cardDrawState = CardDrawState.PLAYERIDLE; //return
                    to IDLE state
                break;
            case REQUESTCARDWAITWMINUSONE:
                //broadcast wprime_i
                DataChainLink last_lcasebigoCardLink =
                    otherPlayersChain.getLastLcaseBigoCard();
                CardVectorRepresentation last_lcasebigoCard =
                    last_lcasebigoCardLink.
                    getCardVectorRepresentation();
                try {
                    last_lcasebigoCard = new
                        CardVectorRepresentation(
                        last_lcasebigoCardLink.Attributes, Deck.
                        numberOfCards);
                } catch (Exception e) {
                    e.printStackTrace();
                    System.exit(-1);
                }
                CardVectorRepresentation w_jCard = new
                    CardVectorRepresentation(last_lcasebigoCard,
                         myPermMatrix, Deck.numberOfCards); //
                            compute w_1

                int valueOfCard = w_jCard.getCardValue(primeNumberZ
                    );
                //modify m-th row
                if (valueOfCard < 0) {
                    System.err.println("error in card value");
                    System.exit(-1);
                }
                //modifynonzerorow
                myPermMatrix.modifyRowNonModuloZ(valueOfCard,
                    primeNumberZ, null); //4b of the algorithm

                //encrypted card from shuffled deck
                EncryptedCard lcaseBigOPrime =
                    myEncryptedVectorDeck.getCardAtPosition(
                    valueOfCard);
                nextLink = new DataChainLink(lcaseBigOPrime,
                    lastSignature, position);
                cardDrawState = CardDrawState.
                    CARDREQUESTWAITWPRIMEN;
                //else same
                break;
            case CARDREQUESTWAITWPRIMEN:
```

```
                System.out.println("Got last link of chain, now I
                    can calulate the card and return the value");
                nextLink = new DataChainLink (new String("Value
                    received"), lastSignature, position);
                cardDrawState = CardDrawState.PLAYERIDLE; //return
                    to IDLE state
            default:
                return null;
            }
        }

        myChain.add(nextLink);
        lastSignature = myCrypto.signString(nextLink.forOutput());
        return nextLink;
    }

    private boolean intializationPhase() {
        boolean endState      = initializationState.equals(
            InitializationState.END );
        boolean croupierBCast = initializationState.equals(
            InitializationState.PLAYERCROUPIERBCAST );
        boolean initializationPhase = ( !endState && !croupierBCast
            );
        return initializationPhase;
    }

    private DataChainLink getInitializationContractionLink() {
        DataChainLink nextLink;
        DataChainLink[] lastPlayerLinks = new DataChainLink[MPGame.
            numberOfPlayers + 1];
        lastPlayerLinks[0] = null;
        for (int playerLink = 1; playerLink <= MPGame.
            numberOfPlayers; playerLink++) {
            if (playerLink != position) {
                lastPlayerLinks[playerLink] = otherPlayersChain.
                    getLastLinkOfPlayer(playerLink);
            } else {
                lastPlayerLinks[playerLink] = myChain.
                    getLastLinkOfPlayer(playerLink);
            }
        }
        nextLink = new DataChainLink(lastPlayerLinks, myCrypto,
            position);
        return nextLink;
    }

    private DataChainLink getWjLink() {
```

```
        DataChainLink nextLink;
        DataChainLink initial_lcasebigoCardLink = otherPlayersChain
            .getLastLcaseBigoCard();
        CardVectorRepresentation initial_lcasebigoCard = null; //
            get w_0 from chain link
        try {
            initial_lcasebigoCard = new CardVectorRepresentation(
                initial_lcasebigoCardLink.Attributes, Deck.
                numberOfCards);
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(-1);
        }
        CardVectorRepresentation nextCard = new
            CardVectorRepresentation(initial_lcasebigoCard,
                myPermMatrix, Deck.numberOfCards); //compute w_1
        nextLink = new DataChainLink(nextCard, false, lastSignature
            , position);
        return nextLink;
    }

    private DataChainLink getWzeroDNC() {
        DataChainLink nextLink;
        int next_u0 = uZeroGen.getNextUZero();
        CardVectorRepresentation cardToSend = myVectorDeck.
            getCardAtPosition(next_u0);
        nextLink = new DataChainLink(cardToSend, true,
            lastSignature, position);
        return nextLink;
    }

    public void receiveNextDNCLink(DataChainLink inputDNC) {
        if (initializationState != InitializationState.END) { //
            player in initialization phase.
            if (inputDNC != null) {
                System.out.println("(" + name + ")received: " +
                    inputDNC.forOutput()); // DEBUG
                if (inputDNC.PlayerPosition != position) {
                    otherPlayersChain.add(inputDNC);
                }

                if ((this.initializationState ==
                    InitializationState.VECTORDECKBCAST)
                        && isCroupier()) {
                    this.dncConcatenation += inputDNC.toString();
                }
            }
```

```
        } else { //card draw
            //GETS DNC LINK AND DECIDES WITCH THE NEXT STATE WILL
               BE

            if (cardDrawState == CardDrawState.PLAYERIDLE) { //
               From Idle State:
                if ( (position == 1) && inputDNC.isLcaseBigOZero()
                   ) {
                    cardDrawState = CardDrawState.PLAYER1RCVW0;
                } else if ( (position != 1) && inputDNC.
                   isWjofPreviousPlayer(position) ) {
                    cardDrawState = CardDrawState.PLAYER1RCVW0;
                } else if ( (position !=1) && inputDNC.
                   isWjprimeofPreviousPlayer(position) ) {
                    cardDrawState = CardDrawState.
                       PLAYERJRCVWJPRIMEMINUSONE;
                } else {
                    //just store the DNC
                }
            } else { // From non-Idle States:
                switch (cardDrawState) {
                case PLAYER1RCVW0:
                case PLAYERJRCVJMINUSONE:
                case PLAYERJRCVWJPRIMEMINUSONE:
                    cardDrawState = CardDrawState.PLAYERIDLE;
                    break;
                case REQUESTCARDWAITWMINUSONE:
                    if (cardDrawBroadCastFlag) {
                        cardDrawBroadCastFlag = false;
                        cardDrawState = CardDrawState.
                           CARDREQUESTWAITWPRIMEN;
                    }
                    break;
                case CARDREQUESTWAITWPRIMEN:
                    if (cardDrawBroadCastFlag) {
                        cardDrawBroadCastFlag = false;
                        cardDrawState = CardDrawState.PLAYERIDLE;
                    }
                    break;
                }
            }
            otherPlayersChain.add(inputDNC);
        }
    }

    /* Private Methods */
```

```
    private boolean received_w_j_prime() {
        //returns true if player received w_j^prime from previous
           player
        if (otherPlayersChain.receivedW_j_primePrevious(position))
           {
            return true;
        } else {
            return false;
        }
    }

    private boolean received_w_j() {
        //returns true if player received w_j from previous player
        if (otherPlayersChain.receivedW_j(position)) {
            return true;
        } else {
            return false;
        }
    }

    private boolean received_w0() {
        if (this.otherPlayersChain.reveivedWZero()) {
            return true;
        } else {
            return false;
        }
    }

    private boolean isCroupier() { //by assumption return true if
       first (position 1)
        return (position == 1);
    }

    private boolean turnToDraw() {
        boolean positionToDraw     = (position ==
           CardDrawerPosition);    //Utilizes 1st assumption
        boolean numberOfCardsDrawn = (cardsDrawn <=
           NumberOfCardsToDraw); //Utilizes 2nd assumption
        if ( positionToDraw && numberOfCardsDrawn ) {
            cardsDrawn++; //side effect!
            return true;
        } else {
            return false;
        }
    }

}
```

6.4.20 implementation_code/java_code/Sha1Signature.java

```
import java.io.UnsupportedEncodingException;
import java.nio.charset.Charset;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

class Sha1Signature {

    public static byte[] getSHA1Byte(String input) throws
       NoSuchAlgorithmException {
        MessageDigest digest = MessageDigest.getInstance("SHA-1");
        digest.reset();
        byte[] sha1;
        try {
            sha1 = digest.digest(input.getBytes(Charset.
               defaultCharset().displayName()));
        } catch (UnsupportedEncodingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            sha1 = null;
        }
        return sha1;
    }

    public static String getSha1(String input) {
        byte[] result;
        try {
            result = Sha1Signature.getSHA1Byte(input);
        } catch (Exception e) {
            result = null;
        }
        return convertToHex(result);
    }
    /*
     * Code originated from here:
     * http://www.anyexample.com/programming/java/
        java_simple_class_to_compute_sha_1_hash.xml
     */
    private static String convertToHex(byte[] data) {
        StringBuffer buf = new StringBuffer();
        for (int i = 0; i < data.length; i++) {
            int halfbyte = (data[i] >>> 4) & 0x0F;
```

```
            int two_halfs = 0;
            do {
                if ((0 <= halfbyte) && (halfbyte <= 9))
                    buf.append((char) ('0' + halfbyte));
                else
                    buf.append((char) ('a' + (halfbyte - 10)));
                halfbyte = data[i] & 0x0F;
            } while(two_halfs++ < 1);
        }
        return buf.toString();
    }
}
```

6.4.21 implementation_code/java_code/Signature.java

```
import java.io.UnsupportedEncodingException;
import java.nio.charset.Charset;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

class Signature {

    public static byte[] getSHA1Byte(String input) throws
       NoSuchAlgorithmException {
        MessageDigest digest = MessageDigest.getInstance("SHA-1");
        digest.reset();
        byte[] sha1;
        try {
            sha1 = digest.digest(input.getBytes(Charset.
               defaultCharset().displayName()));
        } catch (UnsupportedEncodingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            sha1 = null;
        }
        return sha1;
    }

    public static String getSha1(String input) {
        byte[] result;
        try {
            result = Signature.getSHA1Byte(input);
        } catch (Exception e) {
            result = null;
        }
```

```
        return convertToHex(result);
    }
    /*
     * Code originated from here:
     * http://www.anyexample.com/programming/java/
        java_simple_class_to_compute_sha_1_hash.xml
     */
    private static String convertToHex(byte[] data) {
        StringBuffer buf = new StringBuffer();
        for (int i = 0; i < data.length; i++) {
            int halfbyte = (data[i] >>> 4) & 0x0F;
            int two_halfs = 0;
            do {
                if ((0 <= halfbyte) && (halfbyte <= 9))
                    buf.append((char) ('0' + halfbyte));
                else
                    buf.append((char) ('a' + (halfbyte - 10)));
                halfbyte = data[i] & 0x0F;
            } while(two_halfs++ < 1);
        }
        return buf.toString();
    }


}
```

6.4.22 implementation_code/java_code/UZeroGenerator.java

```
import java.util.ArrayList;
import java.util.Collections;

public class UZeroGenerator {
    private int tCardRange;
    private ArrayList<Integer> uZeroValues;

    //public:
    public void dumpUzeroValues() {
        System.out.println("values: " + uZeroValues);
    }

    public int getNextUZero() {
        if (uZeroValues.isEmpty()) {
            return -1;
        }
```

```
        Object obj = uZeroValues.get(0);
        uZeroValues.remove(0);
        int result = ((Integer)obj).intValue();
        return result;
    }

    public boolean removeFromList(int otherUserUZero) {
        Object obj = new Integer(otherUserUZero); //cast
        if (uZeroValues.isEmpty()
                || (!uZeroValues.contains(obj)) ) {
            // error number that other user requests
            // has already been used, or list empty
            return false;
        }
        uZeroValues.remove(obj);
        return true;
    }

    public UZeroGenerator(int cardRange) {
        this.tCardRange = cardRange;
        this.constructUZeroGenerator();
    }

    //private:
    private void constructUZeroGenerator() {
        uZeroValues = new ArrayList<Integer>();
        for (int i=1; i <= tCardRange; i++) {
            uZeroValues.add(new Integer(i));
        }
        Collections.shuffle(uZeroValues);
    }

    public static void main(String args[]) {
        System.out.println("simple demonstation of u0 generation");
        int t = 5;
        UZeroGenerator uZeroGen = new UZeroGenerator(t);
        uZeroGen.dumpUzeroValues();
        int selectedVal = uZeroGen.getNextUZero();
        System.out.println("selected int: " + selectedVal);
        uZeroGen.dumpUzeroValues();
        int otherSelectedVal = (selectedVal + 1 ) % (t + 1);
        if (otherSelectedVal ==0) { otherSelectedVal++; }
        System.out.println("removing: " + otherSelectedVal);
        if (uZeroGen.removeFromList(otherSelectedVal)) {
            uZeroGen.dumpUzeroValues();
        } else {
            System.out.println("Error in removing");
```

```
        }
    }
}
```

6.4.23 implementation_code/java_code/VectorDeck.java

```
import java.math.BigInteger;
/*
 * Represents a deck of cards under the Card Vector Representation
 */
class VectorDeck {
    private CardVectorRepresentation[] vDeck;

    final static String Seperator = ", ";

    public VectorDeck(Deck inputDeck, BigInteger primeNumberZ) {
        vDeck = new CardVectorRepresentation[Deck.numberOfCards +
            1];
        vDeck[0] = null;
        for (int i=1; i <= Deck.numberOfCards; i++) {
            int cardValue = inputDeck.getCardAtPosition(i);
            vDeck[i] = new CardVectorRepresentation(cardValue, Deck
                .numberOfCards, primeNumberZ, null);
        }
    }

    public CardVectorRepresentation[] getVectorDeck() {
        return vDeck;
    }

    public CardVectorRepresentation getCardAtPosition(int i) {
        if ( (i < 1) || (i > Deck.numberOfCards) ) {
            return null;
        }
        return vDeck[i];
    }

}
```

6.5 Appendix E - JIF source code

File listing of JIF Files

6.5.1 implementation_code/jif_code/MPElGamal.jif

```
import java.math.BigInteger;
import java.security.*;
import java.util.Random;
import java.security.interfaces.*;

public class MPElGamal[label L]  {

  private static int __JIF_SIG_OF_JAVA_CLASS$20030619 = 0;

  private final int{L} RandomBitLength = 50;        //randomly
     chosen
  private final int{L} RandomBitLengthLess = 48;    //randomly
     chosen minus two

  private MPKeyPrivate[L]{L} privateKey = null;
  private MPKeyPublic[L]{L}  publicKey = null;
  private BigInteger{L} zPlayer = null;


  //public static int Apos = 0;
  //public static int Bpos = 1;
  //public static int EncryptionArraySize = 2;
  //public static final String Seperator = "-";

    MPElGamal{L; this}() {
        int{L} randomBL = this.RandomBitLength; //saves from side
           effect

        //initialization
        Random{L} rnd = new Random();
        publicKey  = new MPKeyPublic[L]{L}();
        privateKey = new MPKeyPrivate[L]{L}();
        //assignments
        BigInteger{L} biPublic;
        try {
            BigInteger{L} pubP =  new BigInteger{L}(randomBL, rnd);
            publicKey.setP(pubP);
            BigInteger{L} pubG = new BigInteger{L}(
               RandomBitLengthLess, rnd);
            publicKey.setG(pubG);
```

```
            BigInteger{L} privX =  new BigInteger{L}(
               RandomBitLengthLess, rnd);
            privateKey.setX(privX);

            BigInteger{L} pubY =  pubG.modPow(privX, pubP);
            publicKey.setY(pubY);

        } catch (java.lang.NullPointerException npExp) {
            //do nothing
        } catch (java.lang.IllegalArgumentException ilargExp) {
            //do nothing
        } catch (java.lang.ArithmeticException arExp) {
            //do nothing
        }
    }

public MPEncryptedMessage[L] encrypt(BigInteger{L} message) {
    //MPEncryptedMessage[L]{L} result = new MPEncryptedMessage[L]{L
       }();
    return null;
}
}
```

6.5.2 implementation_code/jif_code/MPEncryptedMessage.jif

```
import java.math.BigInteger;

class MPEncryptedMessage[label L] {
    public BigInteger{L} encA; //ElGamal's A
    public BigInteger{L} encB; //ElGamal's B

    MPEncryptedMessage() {

    };
    /*
    MPEncryptedMessage(BigInteger a, BigInteger b) {
        encA = a;
        encB = b;
    }

    public String strOutput() {
        return encA + MPElGamal.Seperator + encB;
    }

    public void add(MPEncryptedMessage summationElement) {
```

```
        encA = encA.add(summationElement.encA);
        encB = encB.add(summationElement.encB);
    }

    public void mutiply(BigInteger c) {
        encA = encA.multiply(c);
        encB = encB.multiply(c);
    }

    public void add(BigInteger a, BigInteger b) {
        encA = encA.add(a);
        encB = encB.add(b);
    }

    public void multiply(MPEncryptedMessage mpEncryptedMessage) {
        encA = encA.multiply(mpEncryptedMessage.encA);
        encB = encB.multiply(mpEncryptedMessage.encB);
    }
    */
}
```

6.5.3 implementation_code/jif_code/MPKeyPublic.jif

```
import java.io.PrintStream;
import java.lang.Object;
import java.math.BigInteger;

class MPKeyPublic[label L] {

  private static int __JIF_SIG_OF_JAVA_CLASS$20030619 = 0;

  public BigInteger {L} p; //El Gamal's p, prime number
  public BigInteger {L} g; //El Gamal's g, random number less than
     p
  public BigInteger {L} y; //El Gamal's y, y = (g^x)mod(p)

  public void setP{L;newP}(BigInteger{L} newP): {L;newP} {
    this.p = newP;
  };

  public void setG{L;newG}(BigInteger{L} newG): {L;newG} {
    this.g = newG;
  };

  public void setY{L;newY}(BigInteger{L} newY): {L;newY} {
```

```
    this.y = newY;
  };
}
```

6.5.4 implementation_code/jif_code/PokerGame.jif

```
import java.io.PrintStream;
import jif.runtime.Runtime;
import java.io.FileOutputStream;
import java.io.BufferedReader;
import java.io.InputStreamReader;

class PokerGame authority (Alice, Bob) {
    public final principal{} p;

    PokerGame(principal{} p) where authority(Alice, Bob) {
        this.p = p;
    }

    public static PokerGame room;

    public static final void main{}(principal{} p, String args[]):{
       p: }
    throws (SecurityException, IllegalArgumentException)
        where caller(p), authority(Alice, Bob)
        {
            Runtime[p] runtime = Runtime[p].getRuntime();
            if (runtime == null) return;
            PrintStream[{}] output = declassify(runtime.stdout(new
               label{}), {});
            if (output == null) return;

            output.println("Starting game");


        MPElGamal[{Alice:}]{Alice:} pmel = null;
        pmel = new MPElGamal[{Alice:}]();
        MPEncryptedMessage[{Alice:}]{Alice:} mpEncMsg = null;

            output.println("Game finished");
        }
}
```

References

[1] http://faculty.washington.edu/moishe/javademos/security/elgamal.java.

[2] Barbara Liskov Andrew C. Myers. A decentralized model for information flow control. 1997.

[3] Aslan Askarov and Andrei Sabelfeld. Security-Typed Languages for Implementation of Cryptographic Protocols: A Case Study. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *Computer Security - ESORICS 2005, Proceedings of the 10th European Symposium on Research in Computer Security*, volume 3679, pages 197–221, 2005.

[4] Aslan Askarov and Andrei Sabelfeld. Security-Typed Languages for Implementation of Cryptographic Protocols: A Case Study of Mutual Distrust. Technical Report 2005-13, Department of Computer Science and Engineering, Chalmers University of Technology and Göeborg University, 2005.

[5] Jordi Castellà-Roca, Josep Domingo-Ferrer, Andreu Riera, and Joan Borrell. Practical Mental Poker Without a TTP Based on Homomorphic Encryption. In Thomas Johansson and Subhamoy Maitra, editors, *Progress in Cryptology - INDOCRYPT 2003*, volume 2904, pages 280–294, 2003.

[6] David Clark. Project description. `http://www.dcs.kcl.ac.uk/staff/david/msc_projects_08.html`.

[7] Caroline Fontaine and Fabien Galand. A survey of homomorphic encryption for nonspecialists. *EURASIP J. Inf. Secur.*, 2007:1–15, 2007.

[8] Shaft Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. *Computer Science Department, University of California - Berkeley.*

[9] Peter Gutmann. Lessons learned in implementing and deploying crypto software. In *Proceedings of the 11th USENIX Security Symposium*, pages 315–325, Berkeley, CA, USA, 2002. USENIX Association.

[10] Fr. Boniface Hicks. Security-typed languages, jif programming, special lecture. `http://www.ece.cmu.edu/~ece732/lectures/18732-jif.pdf`.

[11] Leslie Lamport. Password authentication with insecure communication. *Commun. ACM*, 24(11):770–772, 1981.

[12] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 2001.

[13] Emilie Nodet. Data-flow controll using jif in a heath care system. 2008.

[14] Prof. Rafail Ostrosky. Elgamal's proof of homomorphism, ucla computer science department. `http://www.cs.ucla.edu/~rafail/TEACHING/WINTER-2005/L8/L8.ps`.

[15] Dave King Sandra Rueda Tim Misiak Kiyan Ahmadizadeh Patrick McDaniel, Boniface Hicks. Jif eclipse plugin, penn state university. http://siis.cse.psu.edu/jifclipse/.

[16] Andrei Sabelfeld. Jif exercises, chalmers university. http://www.cs.chalmers.se/Cs/Grundutb/Kurser/lbs/JifLab2006/JifExercises.htm.

[17] Christian Schindelhauer. A toolbox for mental card games. Technical report, 1998.

[18] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, 1996.

[19] Adi Shamir, Ronald L. Rivest, and Leonard M. Adleman. Mental Poker. *The Mathematical Gardner*, pages 37–43, 1981.

[20] K. Vikram Lantian Zheng Stephen Chong, Andrew C. Myers. Jif reference manual, cornell university. http://www.cs.cornell.edu/jif/doc/jif-3.3.0/manual.html.

[21] K. Vikram Xin Zheng Nate Nystrom Lantian Zheng Steve Zdancewic Stephen Chong, Andrew Myers. Jif's home page, cornell university. http://www.cs.cornell.edu/jif/.

[22] Yi Mu Weiliang Zhao, Vijay Varadhara jan. A secure mental poker protocol over the internet.