A Test Bed for Data Hiding in Financial Transactions

Dylan Leigh <dylan.leigh@rmit>

February 2012

RMIT CS&IT Summer Studentship Project
Supervisor: Dr Ron van Schyndel

Abstract

The aim of this project is to develop a framework to allow experiments with data hiding in financial transactions, and for detecting the use of information hiding in financial transactions.
An analysis is made of electronic funds transfer systems in use within Australia, particularly those used for direct credit transactions. A model of financial transaction networks based on Australian systems is developed and a simulator framework which implements the model described is developed for conducting further research. Several example data hiding techniques and scenarios are presented.

Contents

List of Tables

List of Figures

1 Introduction

Much research has been recently been conducted on covert financial networks operating over covert channels [20, 19], but there is little if any research on using standard financial networks as a medium for covert channels. The aim of this project is to develop a framework to allow experiments with data hiding in financial transactions, and detecting the presence of this hidden data.

These covert channels can be used for many purposes including both money laundering and money laundering detection, as shown in the scenarios below (on page 9). Ramos [27] examines a stock/options market trading environment where traders can collude via hiding data in market stock/options price series data while hiding these communications from market regulators. Similar schemes can be used for collusion in other markets, such as auctions; bidders in USA FCC spectrum/bandwidth auctions have used covert channels within the auction process to collude and keep all bids low [25, 26]. A list of possible applications of this technology is listed in section 6.2 on page 42.

1.1 Financial Networks

It is necessary to model individual transactions to add the covert channel or watermark to them. To ensure that the results are applicable to real EFTN, the EFTN used in Australia - under the aegis of the Australian Payments Clearing Association [4] - have been examined. The bulk of the early work on this project was examining the APCA protocol and procedure documentation. The discussion of financial networks is presented in section 2 on page 11.

This "bottom-up" model is a different approach to many existing models used in money laundering which use a graph based flow-of-money model and analyze the graph involved [23].

In particular, high volume, low value, "wire transfer" networks such as the Bulk Electronic Clearing System [7, 8] (section 2.3 on page 13) are examined as these are most applicable to covert channel and watermarking techniques.

Scenario 1: Covert Channels for Money Laundering

Bob is a member of a money laundering network (MLN). He runs a small retail stationary shop. Bob's suppliers for the shop are Sam, Steven and Sally, also part of the money laundering network.

Bob receives money from other members of the MLN who come to his shop to purchase stationary and pay him inflated prices for the merchandise. Bob transfers the dirty money on via inflated prices on his purchases to Sam, Steven and Sally. When Bob makes an order from these suppliers, he transfers the money using the "direct credit" or "wire transfer" service available from his bank.

Sam Steven and Sally need to know who to pass the money on to, so Bob uses the cents value of the bank transfer to encode the the next recipient in the MLN. Amounts ending in 01 cents are transferred to Alice, amounts ending in 02 are transferred to Claire, amounts ending in 03 are transferred to David, and so on.

When Bob makes the transfer determines how much of the transaction the suppliers should pass on to the rest of the MLN. When sent in the first 10 minutes of the hour, half the value of the transaction should be passed on. Between xx:10 and xx:20, a third of the transaction is passed on. Between xx:20 and xx:30, a quarter is passed on. (and so forth).

If every time Bob makes a purchase, he instead sends his suppliers/conspirators a letter or email with the information hidden within, this message may be seen as suspicious. An auditor or police investigator is likely to suspect that these regular messages contain a code. However, the bank transfer is a necessary part of the transaction, and would not raise such suspicion.

Scenario 2: Watermarking for Money Laundering Detection

Banks FOO, BAR and BAZ use the national bulk electronic transfer direct credit networks XYZZY and PLUGH to process and settle bank transfer transactions between their customers accounts. These banks suspect that some of their customers who are sending a high volume of low-value transactions are involved in a money laundering network (MLN). The transfers involved are below the threshold required for the bank to report the transaction to federal anti-money-laundering authorities.

The banks can identify such "suspicious" transactions when sent, and need to be able to somehow "tag" or "flag" these suspicious transactions. The XYZZY system is inflexible and the banks cannot set a "suspicious" flag via the system. The PLUGH network has this capability, but there is a possibility that the suspects have access to PLUGH and can see if an account in their MLN has transactions flagged "suspicious". Thus the banks need to find another way to mark these transactions within their transfer networks.

Due to the very high volume of transactions involved cannot justify the resources[a] to set up their own out of band anti-money-laundering communications network to notify the other banks of the suspicious transaction. Instead of creating such a separate out of band network, the banks agree to add a of watermark to these suspicious transactions, using the transaction data within XYZZY or PLUGH itself; then they can see if an account is receiving transactions from a suspect account.

PLUGH uses C time_t [31] to store transaction times - a digital watermarking algorithm is applied to the timestamp which alters the timestamp by less than a minute, which the suspects are unlikely to notice. The banks examine each transaction received via PLUGH to determine if the watermark is present in the timestamp.

Unfortunately the older XYZZY system only transfers the date, which is not fine enough to be perturbed without the criminals noticing the change. The banks decide that when a suspicious transaction is added to the XYZZY network destined for one of the other banks, they will split the transaction (with the ratio of the split determined by a watermarking algorithm) and recombine the transactions into one before entering it on the customer's statement.

[a] This is a theoretical scenario where banks are not making large profits and have relatively limited resources. The banks would not profit from shutting down the MLN, and would lose the criminals as customers, so cost-effective solutions to money laundering detection are preferable.

1.2 Terminology, Concepts and Abbreviations

APCA Australian Payments Clearing Association - primary operator of EFTN in Australia. See 2.2 on the facing page.

Bank Where the term bank is used, it may refer to any provider of banking services (even those which do not legally fit the legal definition of bank) or any financial institution.

BSB Bank, State and Branch Code. This is a 6 digit code used widely in Australia. The first 2 digits uniquely identify a FI; the third digit identifies the state or territory of the branch, and the last 3 identify a specific bank branch in that state or territory. Accounts must be held at a specific branch, so the BSB code is usually a prefix to an account number.

Clearing The process of settling a transaction, once it has been committed to. The receiver of the funds is generally unable to draw or make use of them until cleared.

COIN "Community of Interest Network" - lower-layer infrastructure system operated by APCA. See section 2.8 on page 21.

CPN "Common Payments Network" - comprises the APCA, RBA and SWIFT lower-layer infrastructure systems. See section 2.8 on page 21.

CS Clearing System; may refer specifically to the upper-layer clearing systems which are used by APCA members (CS1,CS2, etc). See section 2.2.

"deferred net settlement" see section 2.2 on page 12.

EFT Electronic Funds Transfer.

EFTN Electronic Funds Transfer Network(s).

FI Financial Institution. This may include any institution with a direct or indirect (via association with a directly connected FI) interface to an EFTN.

POS Point of Sale.

RBA Reserve Bank of Australia.

Settlement The final transfer of funds/securities from one FI to another.

2 Electronic Funds Transfer Networks

Internationally, the predominant electronic funds transfer network operator is the Society for Worldwide Interbank Financial Telecommunication (SWIFT) [17, 1]. Within Australia the predominant EFTN operator is the Australian Payments Clearing Association (APCA) [4]. The payment and clearing systems they operate are discussed in this section.

2.1 Terminology

Section 1.2 on the facing page explains many of the concepts and abbreviations used in this section.

2.2 Australian Payments Clearing Association (APCA)

APCA operates five payment systems (aka "Clearing Systems"), focused on a particular type of transaction:

- Australian Paper Clearing System (APCS) (CS1)
  - For cheques and similar items (e.g. money orders)
- Bulk Electronic Clearing System (BECS) (CS2)
  - For direct debit and direct credit transactions
- Consumer Electronic Clearing System (CECS) (CS3)
  - For point of sale transactions (e.g. ATM and EFTPOS).
- High Value Clearing System (HVCS) (CS4)
  - For special high value, real-time payments (generally related to investments and trading).
- Australian Cash Distribution and Exchange System (ACDES)
  - For distribution of cash - notes and coins - between banks and branches.

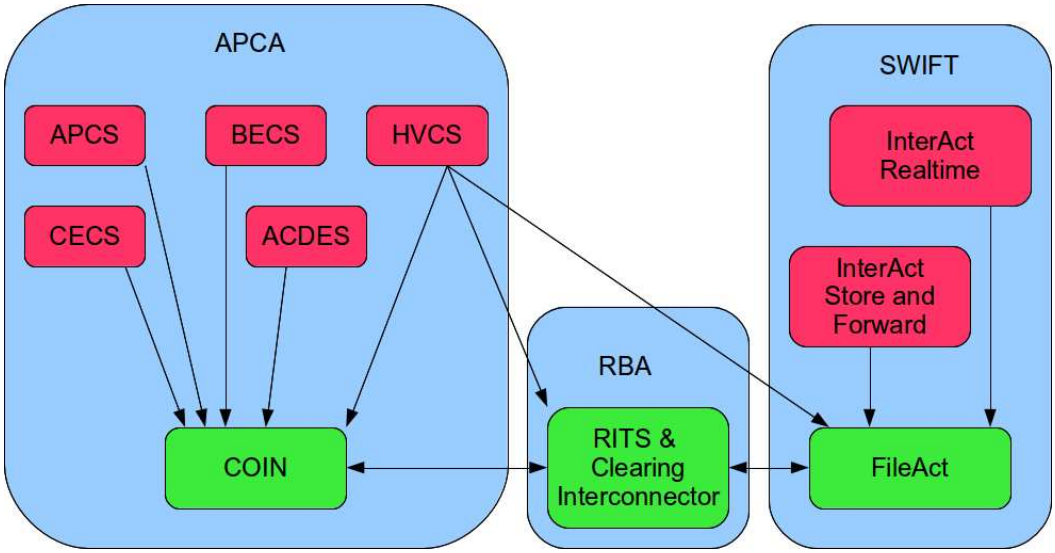

Figure 1: Overview of APCA and SWIFT networks

These five payment networks - each discussed in the following sections - operate by transferring "files" over a lower layer referred to as "infrastructure system". The "files" each contain records on individual transactions.

APCA operates one infrastructure system (known as COIN), the Reserve Bank of Australia operates a "Clearing Interconnector" which interfaces with the international SWIFT "FileAct" system. These three systems (COIN, FileAct and the Clearing Interconnector) comprise the "Common Payments Network". These "lower layer" infrastructure systems are described in section 2.8 on page 21.

Many details of these systems are classified and are only available to members of APCA (i.e. the financial institutions connected to the systems). Access to the classified sections was unavailable when preparing this report, however some classified details are possible to determine from references in the unclassified sections.

Most of the Clearing Systems operate on a *deferred net settlement* basis - transactions between two financial institutions are pending (and the proceeds of the transfer unavailable to the recipient) until the FIs accounts are settled. The net total of transfer of value between the two FIs is determined (based on the totals of all transfers from the prior business day) and the reserve account of the FIs are adjusted appropriately. This is typically performed by the next business day; most systems specify times outside of business hours where settlement of transactions on the previous business day is performed.

Note that members must have bilateral agreements with each other to take part in most interactions (i.e. even if two banks are both connected to the system, the banks must have a specific, preexisting agreement to process transfers between their customers). The regulations of each clearing system specify the minimum requirements of such agreements. See section 2.8 on page 21 for further information.

2.3 Bulk Electronic Clearing System (BECS)

The Bulk Electronic Clearing System (BECS aka CS2) [7, 8] is a clearing system intended for bulk electronic exchange of payment instructions. This makes it suitable for many different types of payments. In practice, this system is used for direct credit ("wire transfer") and direct debit payments.

BECS is designed for batched, high volume transactions. As of May 2011, direct credit payments average 4.7 million transactions per day, with a value of $23.7 billion per day [4].

Like APCS and CECS, it operates on a deferred net settlement basis and can be used as a backup for other clearing systems which also operate on the same basis. BECS can handle payments up to 100 million dollars.

Financial institutions connected to the network may process transactions for an associate which is not a full member of the network. Members may "represent" other members or non-members in transactions. There are multiple tiers of membership in this manner. According to [8], entrance fees for Tier 1 Participating Membership are $94,000; for Tier 2 Participating Membership, $5,000. A variety of operational fees also apply according to formulae laid out in the regulations.

The BECS Procedures specify times of day during which files are exchanged between FIs (based on Sydney time). Files may also be exchanged at any time between particular FIs by agreement between those FIs. FIs are required to ensure that during any exchange of files all files available for exchange at that time are exchanged.

Files must be in EBCDIC format and are normally exchanged via the COIN infrastructure system (see section 2.8 on page 21), encrypted with Triple DES "or equivalent"; if COIN is unavailable, files can be exchanged via email messages not exceeding 5MB (with specific security considerations applied to the messages), or 600MB CDROM discs.

Settlement of the transactions which are described in the files must be completed by 9am Sydney time the next business day. The inward/outward credit/debit values for each FI must be calculated and submitted to the system for collation by 11pm Sydney time on the day of exchange. Collation is

Record Type	Record Type
0	Descriptive
1	Detail
2	Returned Item
3	Refusal
7	File Total

Table 1: BECS Record Types

performed on a regional and national level, and include subsidiary members in a lower "tier".

While many of the specific details of the files and records are classified, there are five types of records used in BECS files (table 2.3)

2.3.1 Descriptive Records

The descriptive records are common to all files; although details are classified it is stated that "Field Descriptions" vary between file types. It appears that descriptive records are header records which describe the fields contained in the other records.

BECS members are not required to reject records which do not conform to the "standard" field specifications.

2.3.2 Detail Records

The exact details of the record types used in BECS exchanges are confidential and not available to us. The "Detail" records are of several types, one of which is used to define Credit and Debit transactions.

The exact listings of transaction file records is classified, although it can be determined through unclassified procedures and regulations that certain elements exist. Table 2 on the next page describes these elements.

It should be noted that although the destination account name is not only optional, but even if it is sent the receiving FI is under no obligation to use it to verify that the destination account is correct. The destination account ID is the only key used to identify the destination account for the transfer. The legal requirements of the sending and receiving FIs are detailed in section 4.18 and 4.19 of the BECS Procedures [7].

As it is intended for "direct entry" payments BECS has simple requirements. Although not mentioned in the unclassified text, it can be assumed that

Field	Required	Notes
Sending FI ID	Yes	Typically convertible to/from BSB
Source Account ID	Yes	With BSB if applicable
Destination Account ID	Yes	Only key used for determining destination
Destination Account Name	No	Not required to be used for verifying destination
Transaction Type	Yes	See following table
Description	No	Customer specified; limited character set

Table 2: BECS Transaction Record Fields

Code	Transaction Type
13	externally initiated Debit Items
50	externally initiated Credit Items (except those using codes 51-57)
51	Australian government security interest
52	basic family payments/additional family payment
53	pay
54	pension
55	allotment
56	dividend
57	debenture/note interest

Table 3: BECS Transaction Types

transactions store the dollar value of the transfer; it is also likely that a transaction ID number is also used.

2.3.3 Return, Refusal and Summary Files

The details of return and refusal records are almost fully expunged from the publicly available documentation. The procedures for these operations are not available. One detail is known: there is a set of "Reverse" FI ID numbers - the BSB associated with the FI with the prefix "997" - which is used as the FI ID in reversal operations.

A summary file must accompany each normal file. The details of the summary files are confidential; it is implied that these summary files are used for collation and verifying settlement.

2.3.4 Further Details

The BECS Regulations and Procedures contain many instructions which are not relevant to our model; and some which are not currently considered but

may be relevant later. These include:

- Procedures and regulations for the identification and authentication of users with access to BECS.
- Special procedures and file exchange times for files relating to government transactions.
- Procedures for unusual events, such as:
  - Handling incorrect or invalid account numbers
  - Resolving settlements if the credit/debit values for two FIs do not match at the 11pm collation time.
  - Failure of two FIs to settle.
  - Insolvency events - when a financial institution is not able to settle the transaction because the account or the institution has insufficient funds to transfer.
- Interest adjustments.
- Prudential considerations.

These situations may be considered in future versions of the model but are not considered at this time.

2.4 Australian Paper Clearing System (APCS) (CS1)

The Australian Paper Clearing System [5, 6] is a deferred net settlement clearing system primarily intended for clearing payment made via "paper drawings" - cheques (including personal cheques, bank cheques and Travellers' cheques), money orders, and similar paper credit items.

These paper credit items are referred to simply as "Items" in the APCS documents. The physical or virtual representation of the Items may or may not be transferred independently of the funds, which are settled between banks on a net basis. Handling of the Items in this manner is a key feature of APCS which differentiates it from BECS and CECS.

Like BECS and CECS, it operates on a deferred net settlement basis and can be used as a backup for other clearing systems which also operate on the same basis. Exchange times are laid out in the procedures and there are multiple tiers of (directly connected) participating members and associate members. Collation and settlement

APCS includes extensive procedures and schedules for:

- Magnetic Ink Character Recognition systems for cheque printing and reading.
- Handling and delivery of the physical paper Item.
- The electronic transmission of value.
- Presentment (identifying or exhibiting the Item to the drawee financial institution [2, section 62]) and locations where presentment is to take place.
- Determining if the cheque (or other Item) should be honoured.
- Procedures to implement for the dishonouring of items.

Many of these procedures are in place to comply with the commonwealth Cheques Act 1986 [2].

Complex procedures apply for the schedules of Item shipping and value transfer; these procedures vary somewhat between particular organizations and tiers, and a standardized flow chart format is proscribed in the APCS regulations to document these schedules. However, most of the actual detail of the procedures is confidential.

2.4.1 Electronic Presentment & Dishonour Holiday Calendar

Most of the detail of the APCS procedures is confidential. The only record format which is publicly available is used for transmitting a calendar of holidays (days which are not counted during the schedule for processing Items) to financial institutions. The files use one header record, many detail records, and one trailer record, similar to the BECS transmissions. Details are in Appendix J of the APCS procedures [5].

2.5 Consumer Electronic Clearing System (CECS) (CS3)

The Consumer Electronic Clearing System[11, 12] is intended primarily for processing transactions from point of sale devices (including ATM and EFTPOS transactions). CECS can also be used for processing paper-based payment instructions which are not covered by APCS.

From January 2011, EFTPOS transaction handling is transitioning to a new EFTPOS-only system called “ePal”. CECS still determines the security requirements of devices which handle EFTPOS cards and is still capable of handling transactions during the transition period

CECS uses specific terminology to refer to the participants in a POS transaction:

Cardholder: Customer who has been issued a card with which to conduct transactions. This may be a credit or debit card, prepaid or linked to a bank account.

Merchant: Person or organization which delivers goods or services to a cardholder at point of sale (POS).

Issuer: Organization which issues a consumer with a card (e.g. the consumer's bank).

Acquirer: Organization which processes transactions for a POS device (e.g. bank operating an ATM or providing an EFTPOS Terminal service to a merchant).
It is the acquirer's responsibility to discharge the responsibilities of the Issuer to the cardholder.

Terminal: Device which the cardholder (or in some cases, a merchant) uses to make the transaction request; includes a Pin Entry Device (PED).

Procedures vary depending on the situation of withdrawal or transfer; if it is an ATM withdrawal or a merchant must be paid; if the card is credit, debit or prepaid; and differing relationships between the Acquirer, Issuer and Merchant (some of whom may be the same organization).

The standard interchange involves a number of transaction types (see table 4 on the facing page) . These are implemented using a standard set of message types (see table 5 on the next page). Note that "request" messages are from Acquirer to Issuer and "response" messages are from Issuer to Acquirer.

The network management and key management protocols are not examined in detail in this project.

It can be seen from table 4 that most transactions involve a "Financial Transaction Request" message pair, so this message will be examined in more detail. Unlike BECS and APCS, the CECS documentation makes publicly available the details of the protocols, including the character-level format of the messages.

The format is documented on page 91 of the CECS Manual [11]. Several fields are of particular interest to this project:

1. Numeric values are typically sent in a textual representation.

Transaction Type	Messages (in order)*	Notes
Pre-authorized Transaction	0100, 0110, (0420, 0421, 0430),0220,(0221),0230	
Balance Enquiry Transaction	0200,0210,(0420,0421,0430)	
Purchase Transaction	0200,0210,(0420,0421,0430)	
Cash Withdrawal Transaction	0200,0210,(0420,0421, 0430),(0220,0221,0230)	ATM or EFTPOS
Combined Purchase and Cash-Out	0200,0210,(0420,0421,0430)	EFTPOS only
Fall-Back Transaction	0220,(0221),0230,(0420,0421,0430)	(see below)*
Refund Transaction	0200,0210,(0420,0421,0430)	
Reconciliation Transaction	0520,(0521),0530	(see below)*

Table 4: CECS Standard Transaction Set

- Brackets indicate optional messages (for errors or unusual events).
- Pre-authorized transactions are used when the final cost of transaction is not known before purchase (e.g. fuel dispenser, pay phone).
- Fall-back transactions are used in some specific cases when an EFTPOS Transaction has failed.
- Reconciliation transactions are used to collate and confirm the number and value of transactions processed by a node.

Request	Response	Description
0100	0110	Authorization Request
0200	0210	Financial Transaction Request
0220	0230	Financial Transaction Advice
0221	0230	Financial Transaction Advice Repeat
0420	0430	Acquirer Reversal Advice
0421	0430	Acquirer Reversal Advice Repeat
0520	0530	Acquirer Reconciliation Advice
0521	0530	Acquirer Reconciliation Advice Repeat
0800	0810	Network Management Request
0820	0830	Network Management Advice

Table 5: CECS Standard Interchange Message Types

2. The "Systems trace audit number" is a 6 digit ID which uniquely identifies a transaction for a given terminal and day.

3. Transaction amount is stored in textual format, with 10 digits of dollars and 2 digits of cents.

   (a) Separate amount fields exist for a "cash component" (for EFTPOS cash out) and "Transaction Fee".

4. Transmission date and time as well as local (transaction) date and time are stored in the message with precision to the second. Year is not stored.

5. The cardholder account is identified directly by account number (variable length, at most 13 digits).

6. No merchant account number is used; instead a "Card Acceptor Identification Code" (15 characters) and "Card Acceptor Name/ Location" (40 characters) are used to identify the merchant and location.

7. The Issue and Acquirer are identified by their institution code.

8. A Settlement Date is stored which is the date the Acquirer expects to reconcile the accounts.

The form and format of the data in the Financial Transaction Request will be useful in determining what data is stored in our model and how data can be perturbed to form a covert channel.

Much of the CECS regulations concern the security of the transaction; security of terminals and their components, particularly Pin Entry Devices; transmission of sensitive data; minimum standards for encryption and key management; and security auditing requirements.

CECS also has extremely detailed regulations for the operations of terminals (including ATMs); including their form and layout, operations which the cardholder can perform (e.g. checking their account balance), and physical and on screen signage, including notification of fees.

2.6 High Value Clearing System (HVCS) (CS4)

The High Value Clearing System [9, 10] is used to make time-critical, high value transfers of funds, primarily for financial market trading purposes. HVCS transfers are usually between FIs themselves rather than customers,

although occasionally transfers are made on behalf of a customer. 0.1% of APCA network transactions are via HVCS, but they comprise 70% of the total value transferred each day[4].

Despite the name the HVCS is not necessarily for large amounts, there is no minimum on the amount transferred in a transaction. HVCS is characterized by low volume, time sensitive (usually real-time) and irrevocable transactions.

HVCS is more tightly integrated with SWIFT and RBA systems, in order to provide real time settlement and irrevocable transfer. Unlike the previously discussed deferred net settlement clearing systems, during operating hours, transactions are immediately processed and funds immediately transferred.

Transactions are usually final, and can only returned or recalled in the case of special Warehoused or Future Dated payments.

If real time settlement is not available due to technical difficulties, deferred settlement may be used. Payments marked as real time or "today only" are rejected if they cannot be settled within the permitted time.

Although very different in operation to APCS, BECS and CECS, HVCS may be used as a backup for settlement of transactions for the other systems if they are inaccessible, and vice versa.

Access to HVCS is more limited than to the other APCA clearing systems. As our project is concerned with high-volume financial transactions, and the HVCS is a low volume system, it is not examined in detail.

2.7 Australian Cash Distribution & Exchange System (ACDES)

The Australian Cash Distribution and Exchange System [13, 14] is used to manage the exchange of cash (notes and coins) between banks to ensure that banks have an adequate supply of cash and change on hand to conduct business. It is also used to track and audit cash movements.

As this is an inter-bank system with no interaction with regular customers (consumer or business) it is not examined in detail.

2.8 COIN and CPN Infrastructure Systems

Infrastructure Systems are the "lower" layer used to transfer the files containing the "upper" layer clearing systems. The primary infrastructure system used by the APCA systems is COIN.

Community of Interest Network (COIN) was established by APCA in 2009. It is also known as IS1 (Infrastructure system 1). By August 2011, 90% of APCA systems have been migrated to COIN [4].

COIN members are only permitted to use the network to exchange approved traffic, and only with other members who the member has made an Agreement to Exchange. The forms of these agreements are laid out in the regulations for each clearing system (APCS, BECS, CECS, HVCS and ACDES) and include minimum terms (e.g. indemnifying the other member from liability in various situations).

Members are not obliged to engage in any business with any other members even if they are part of the same network. In some cases, one member may act as an intermediary for two members who do not have a bilateral agreement.

COIN is part of the CPN (Common Payments Network), which includes the RBA Clearing Interconnector and the SWIFT FileAct service[18]. FileAct is the "lower" layer for the worldwide SWIFT payment networks. Both FileAct and COIN use the ISO 20022 standard for Financial Services Messaging [16], which supersedes the ISO 15022 standard [15] previously used for SWIFT networks.

Each individual link within COIN is via an IPSec VPN connection between the two COIN members. COIN specifies approved times for network maintenance and downtime; members are expected to maintain their connection to the network at all other times. Minimum standards are specified for host security, encryption, digests, key management, auditing and incident management.

2.9 SWIFT Networks

The Society for Worldwide Interbank Financial Telecommunication (SWIFT) [17] operates the largest and most comprehensive international financial network. Their infrastructure system is the SWIFT FileAct service[18]discussed in the previous section.

"SWIFTNet InterAct Realtime" is a real-time gross settlement system similar to APCA HVCS. "SWIFTNet InterAct Store and Forward" is a deferred settlement system similar to the other APCA systems. These networks are more general purpose than the APCA networks.

Like the APCA networks, SWIFT requires that members have bilateral agreements (with certain terms of the agreement regulated) before they exchange transactions. Members are not obliged to do business with another member that is part of the same network.

SWIFT does not operate accounts for any members, and it does not take part in clearing or settlement.

2.10 Summary

We have identified several categories of EFTN, their operation and some transaction metadata information:

1. Deferred Net Settlement
    (a) Bulk Direct Credit / Direct Debit (including Wire Transfer, Payroll/Salary, Bill Payment)
    (b) Point of Sale (ATM/EFTPOS/Credit Card)
    (c) Paper Credit (Cheques/Money Orders)
2. Realtime Gross Settlement
    (a) Financial Trading / Investment

All of these EFTN operate over an Infrastructure System which is responsible for transferring files between the FIs. The records within these files are formatted according to the "upper" layer EFTN.

BECS (the bulk direct credit EFTN) processes 4.7 million transactions per day, with value of $23.7 billion per day, between its 65 top tier members[4]. The average transaction value is thus calculated as approximately $5042.7, with each top tier member processing approximately 72308 instructions per day.

3 Model

Our model, based on analysis of the APCA payment systems, consists of 4 layers:

1. The **Infrastructure Layer** specifies the general format and transmission of files, to allow them to be inter-operated by the other layers.
2. The **Operations Layer** specifies operations which can be performed on records and files.
    - Operations may be relatively high level (e.g. rejecting a transaction; settling an exchange between institutions) or low level (e.g. quantization of a field, sorting a set of records).
3. The **Protocol Layer** specifies a number of financial network protocols. Protocols are defined in terms of their (layer 2) operations.
4. The **Node Layer** determines the parties involved in the simulation by defining a set of nodes (banks, customers, etc) and the other nodes they interact with (using one or more protocols defined in layer 3).

The "lower layer" infrastructure systems used in EFTN correspond to layer 1; the "upper" layer EFTN systems correspond to layer 3. Layers 2 and 4 are simulation program constructs and not representative of any particular component of a real EFTN.

3.1 Infrastructure Layer (1)

The infrastructure layer defines the format and encoding of files and how the files are transmitted. It is the task of this layer to allow the upper layers to inter-operate. Layer 1 corresponds to the "lower layer" infrastructure systems used in EFTN networks.

3.1.1 Implementation

The infrastructure layer used in the simulator from this project defines the following rules:

- Files are stored in CSV (comma separated value) format.
    - One line per record or transaction.

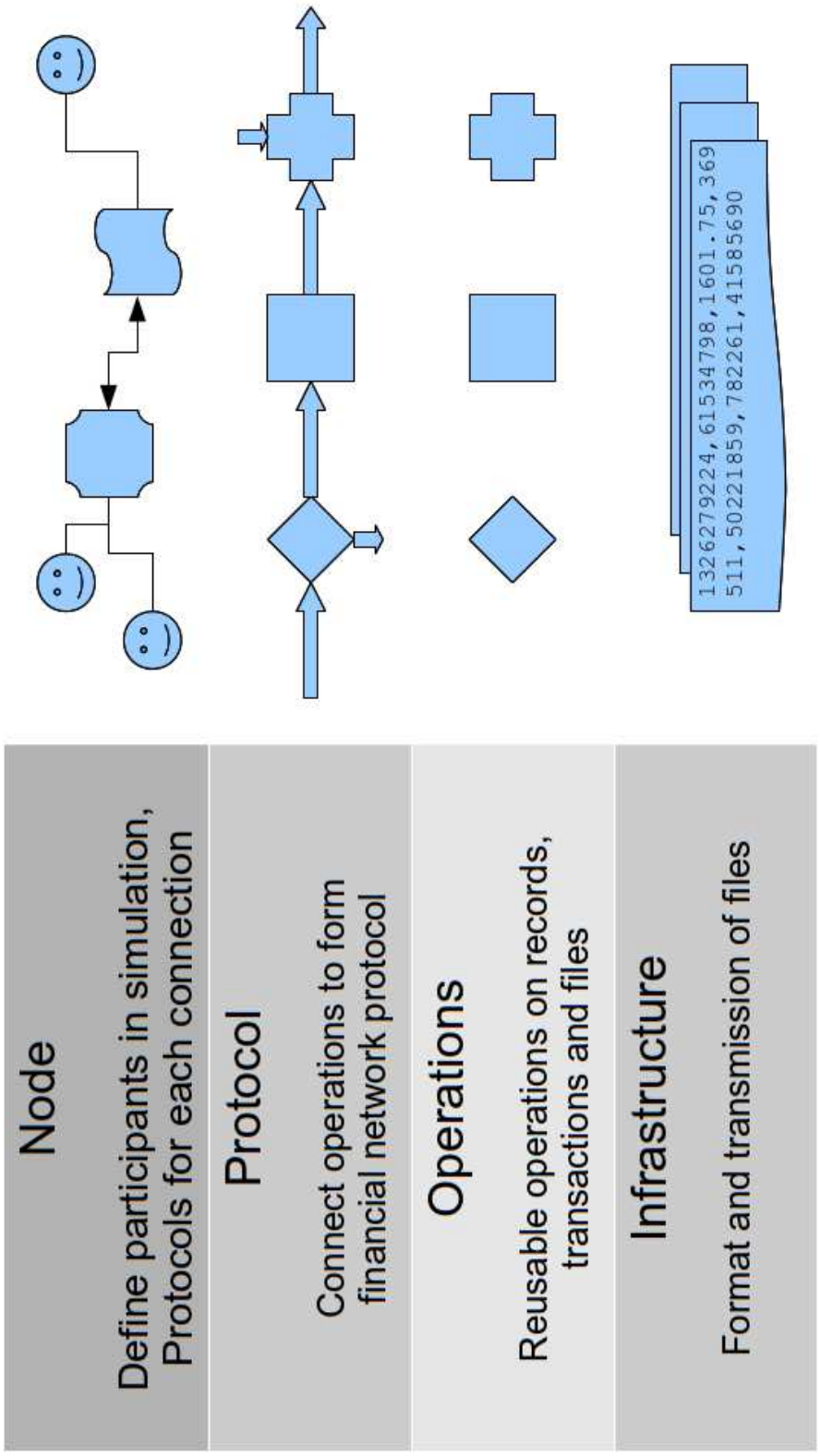

Figure 2: Overview of 4 Layer Model

  - The first line of a file or transfer (for pipe/socket connections) should be a descriptive header which can be used by the software to verify the format of the input.

- Records are stored in ASCII form, including numeric data.
- Timestamps are stored in C time_t format [31] (seconds since January 1, 1970) for compatibility.
  - Depending on the protocol, timestamps may be quantized to less precision.
- Files may be transferred without wrapping or framing (other than the CSV header) via any IPC or network protocol, including pipes, sockets, TCP or HTTP.
  - Upper layers should determine how their operations communicate, including defining port numbers and addresses if required.

These rules ensure that the simulator data can be easily read by human or computer, stored on disk, transferred via a variety of IPC mechanisms and used by software on multiple platforms.

3.2 Operations Layer (2)

The Operations Layer defines a set of operations which may be performed on sets of transactions or records. Each operation is a program, which accepts sets of input transactions/records and (usually) produces an output set.

Some operations may be very general (e.g. sort) and some may be specific to a protocol (e.g. a settlement operation). Some operations (such as sorting and filtering) may be performed for protocol or visualization purposes.

Operations may require additional input (such as the field to sort by, or data to hide). In some cases the operation must produce additional output (e.g. deferred net settlement, the operation must adjust the FI reserve accounts).

3.2.1 Example Operations

- Sorting a set of transactions (e.g. by time of transmission).
- Generating a set of random transactions.

- Determining if transactions within a set should be accepted or rejected (and generating appropriate accept/reject records).
- Filtering the set of transactions (e.g. for a specific destination FI)
- Hiding data within a transaction set.
- Examining a transaction set to determine if a set contains hidden data.
- Collating transaction data for deferred net settlement.
- Clearing settlement on a set of transactions.
- Quantization of a field (e.g. timestamps)

3.3 Protocol Layer (3)

The protocol layer is responsible for modelling a EFTN protocol, which messages are sent from one party to another in order to process transactions. Protocols are implemented by connecting operations together.

For example, a protocol for real time, immediate settlement (similar to APCA HVCS - see on page 11) would only have one operation to transfer and settle a transaction. A protocol which uses deferred settlement may have operations to accept pending transactions, total the accounts, and apply the funds transfer at the end of the day.

There may be any number of protocols operating at this layer. A covert channel may span multiple EFTN protocols. These protocols may interact with each other, and in some cases transactions may be transferred between alternate nodes using alternate protocols.

Like the APCA EFTN protocols, modelled protocols may have "tiers" with nodes at different tiers using a variation of the protocol. Some operations (such as splitting, merging, aggregating and clearing) may have particular significance at the boundary of each tier.

3.3.1 Example Protocol

Table 6 on the next page shows an example protocol involving two financial institutions and one clearing house, and the operations which are performed at each step.

Step	Sending FI	Clearing House	Receiving FI
1	Generate Transactions		
2		Filter	
3			Accept/Reject
4	Notify (rejected transactions)		
5	Collate (accepted transactions)		
6		Settle funds	
7			Clear transactions

Table 6: Example Protocol

3.4 Node Layer (4)

A node is a distinct individual or organization within the simulation. The node layer is responsible for specifying these nodes and how they interact (i.e. which protocols they use between each other).

Nodes may represent any sort of financial institution: banks of any size or tier, interbank network operators and clearing houses, as well as account holders and customers who initiate and receive transactions.

The nodes in a simulation and their behaviour when initiating transactions may be defined manually or generated procedurally. For example, in a simulation involving a bank with millions of individual customers, those customers might not be individually simulated as nodes; instead the bank node runs an operation which generates many random transactions during the simulation. Alternatively, a large sample set of transactions may be pre-generated before the simulation. This is much simpler than modelling each customer as an individual node.

The kind of transactions generated and accepted, the connections to other nodes, and the protocols used must all be specified. This includes traffic which is transferred on to another node (e.g. a Tier 1 bank accepting direct credit transfers for a Tier 2 associate). Nodes may have differing visibility to transaction data (e.g. a clearing house may see BSB numbers but not account details; a customer may not see some "internal" protocol data).

In some simulations it may also be necessary to note which nodes are "friendly" or "unfriendly" i.e. which nodes are taking part in a money laundering detection network, which nodes may be part of a criminal organization, and nodes which are not involved in either.

4 Simulator

4.1 History

A prototype version was developed in Python [30] which used a hardcoded algorithm ("split-amount", see section 5.3 on page 39) and contained only the transaction generator, a "split" and a "join" program. Once the concept and program was shown to be feasible the existing modular simulator was then developed which is documented in this section.

The latest version of Python on the RMIT CS&IT Unix servers is 2.6. This meant that some sections - written on a desktop computer with a later version - had to be rewritten for Python 2.6. Deprecated string formatting syntax was used, and the now deprecated *optparse* argument parser was used instead of the more recent *argparse* module.

The simulator source code can be found in Appendix B on page 53. During the latter stages of development the source code of the simulator shrank in size considerably due to increased modularization, increased use of library modules and refactoring of code into the Transaction class.

4.2 Usage

The simulator programs each perform one operation; they must be connected together to form a protocol, which may connect multiple nodes. For example, a bank may receive transactions from several sources, remove some from the network for processing, send some onwards to other banks untouched, perform steganographic procedures on others, etc. These operations would be simulated by one or more individual programs.

Output from each program is displayed on standard output, and all programs can take any number of files, sockets or pipes as input (using "-" to specify stdin input) specified on the command line.

Intermediate files can be used to keep records of experiments, or shell input/output redirection can be used to connect programs together without the use of intermediate files. The standard Unix utility "tee"[32] can be used to direct output to standard output while also piping to another program.

Command line options and arguments for each program is explained in section 4.4 on the following page.

Appendix A on page 50 contains several example sessions using the simulator.

4.3 Transaction CSV Format

The current simulator uses a simplified transaction record format. Each transaction is represented by a single record, one line of comma separated fields. The Transaction Generator subsection (on this page) describes this format.

4.4 Simulator Programs (Operations)

There are six programs which form the core of the simulator, each described in this subsection. The simulator is designed around the exchange of files similar to the APCA EFTN protocols.

Note that some programs may execute another as part of a complex operation. Some programs can also be used for presentation of data (e.g. to sort and filter transactions) as well as constructing protocols.

4.4.1 Transaction Generator

Usage: `trans-gen.py <number>`

`<number> = number of transactions to generate.`

The Transaction Generator (trans-gen.py) outputs a CSV (comma separated value) stream containing a random set of transaction records. The fields of each record conform to the format in Table 7 on the next page.

All values are randomly generated linearly between their maximum and minimum except Amount. The calculation of amount is discussed in detail in section 4.4.1.1 on page 32.

It is likely that the volume of transactions differs during different times of the day (business hours in particular) as well as on different days of the week, days of the month and periods of the year (e.g. approaching Christmas, taxation dates, end of financial year). Determining an accurate model for how date and time affect the volume or amount of transactions is beyond the scope of this project; as such, the generator currently generates random time for transactions linearly between the maximum and minimum.

The length of in transaction ID (9 digits) and account numbers (8 digits) are those used by the Commonwealth Bank [3]. The ABA bank payment format allows at most 9 digits for account number, and the CECS Financial Transaction Request message allows up to 13 digits [11].

The description and destination account name fields are left blank by the transaction generator. Using these language specific fields is beyond the scope

Field	Type	Minimum	Maximum	Notes
Timestamp	integer	1325336400*	1328014800*	time_t [31]
Transaction ID	integer	1	999,999,999	9 digits
Amount	Decimal*	1	1,000,000	2 decimal places*
Source Account BSB	integer	0	999,999	6 digits
Source Account	integer	0	99,999,999	8 digits
Destination Account BSB	integer	0	999,999	6 digits
Destination Account	integer	0	99,999,999	8 digits
Account Name	String	(n/a)	(n/a)	(unused)
Customer Description	String	(n/a)	(n/a)	(unused)

Table 7: Transaction Generator output data

- Decimal [29] is a Python fixed and floating point class with guaranteed precision suitable for financial data.
- The transaction generator rounds the Amount to 2 decimal places (i.e. integer cents), although the format and programs will handle amounts with fractional cents correctly.
- The Timestamp minimum and maximum are equivalent to midnight (00:00:00) on 2012-01-01 and 2012-02-01 respectively.

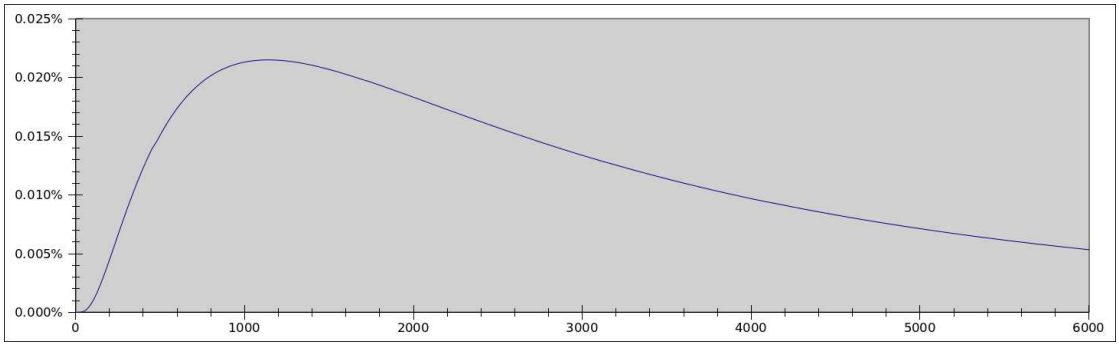

Figure 3: Probability density function of randomly generated transaction amounts.

$$AmountPDF = \frac{1}{x\sqrt{2\pi\sigma^2}}\, e^{-\frac{(\ln x-\mu)^2}{2\sigma^2}} \quad , \mu = 8.0257, \sigma = 1 \tag{1}$$

$$AmountMean = e^{\mu+\sigma^2/2} = 5042.7 \tag{2}$$

$$AmountMode = e^{\mu-\sigma^2} = 1125.2... \tag{3}$$

of this project at this time. Note that the account name is not required by BECS protocols and may not be transmitted by some banks; the regulations do not require the receiving financial institution to check the account name of the receiving account, and indemnify the receiving FI against transfers to unintended recipients as long as the correct account number is used.[8]

4.4.1.1 Amount Calculations

Amount has a log-normal probability density function with $\mu = 8.0257$ and $\sigma = 1$ (see equation 1 and figure 1 on the current page). These values are chosen to generate amounts with a mean average (equation 2) of \$5042.7[1], the same as APCA direct credit transactions in May 2011[4].

The generated figures also conform to "Benford's law" [21, 22]. The first digit of figures in genuine financial transactions follow a power law probability distribution, and the generated digits follow the probable financial digits closely (see figure 4 on the facing page). The difference in the digit 2 is possibly quantization error, or due to too high a frequency of figures in the

[1] The mean of \$5042.7 is calculated using figures of 4.7 million transactions with value of 23.7 billion Australian Dollars; these are the only values which could be obtained from APCA.

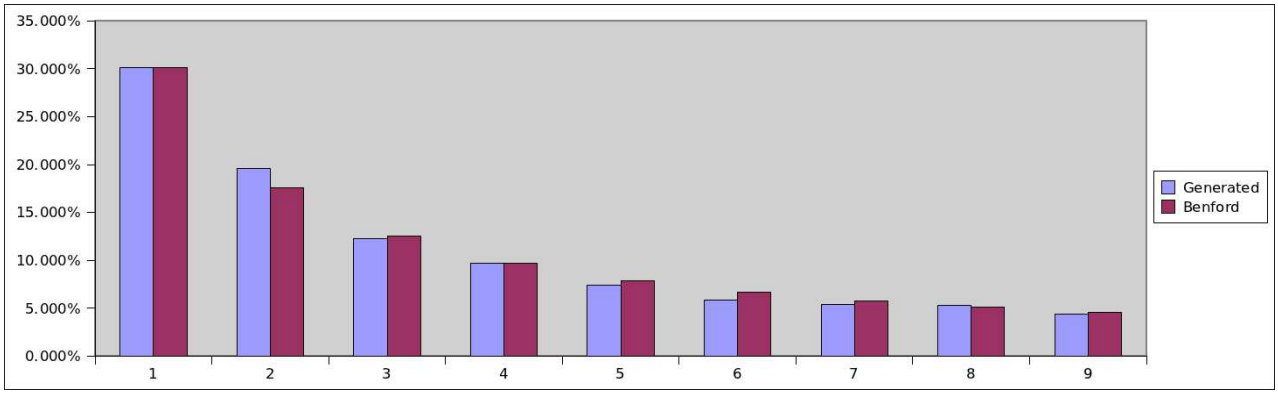

Figure 4: Comparison of first digit of generated amounts with Benford's law probabilities

$2000-2999 range after the peak at $1125. If the latter, the sigma of the generated amount distribution should be adjusted.

Note that values outside the maximum or minimum are discarded - rather than being clamped - to prevent changing the probability of the maximum or minimum values appearing in output.

Despite several attempts real transaction data could not be obtained from any financial institutions. Even anonymized data is confidential and unavailable. The only data available is the APCA monthly average figures (see section 2.10 on page 23).

4.4.2 Insert

The INSERT program takes a set of input transactions and hides information inside them to produce a set of output transactions. The algorithm used to perform the data hiding must be specified (see section 5 on page 37 for details of the algorithm modules). The data to hide is specified as a positive integer on the command line.

Note that the insert program may generate more transactions than it receives, depending on the module used. The hidden code may have size limitations (again depending on the module).

Usage: `insert.py <module> <code> <transactions.csv> [....csv]`

```
<module> is the name of the data hiding algorithm module (without
the .py extension)
<code> is the data to hide.
<transactions.csv> files containing transactions to conceal
```

```
data
(use - for stdin)
```

The source code of the INSERT program is in Appendix B.1.2 on page 55.

4.4.3 Extract

The EXTRACT program takes a set of input transactions and attempts to find hidden data within them. It displays the hidden data when it has analyzed all of the input.

Like the Insert program, the algorithm used to perform the data hiding must be specified (see section 5 on page 37 for details of the algorithm modules). The extract program may or may not display the transactions, depending on the module used (see the module for more details).

Usage: `extract.py <module> <transactions.csv> [....csv]`

```
<module> is the name of the data hiding algorithm module (without
the .py extension)
<transactions.csv> files containing transactions to conceal
data
(use - for stdin)
```

The source code of the Extract program is in Appendix B.1.3 on page 57.

4.4.4 Sort

The SORT program sorts the input transactions according to a specified key field.

Multiple options may be specified; the input will be sorted repeatedly in the order in which the arguments appear (this means that the primary sort key must be specified last, and lesser keys first). The sort is stable (i.e. order is preserved for items with the same key value).

Note that the order of transactions or records may be significant as it can be used as a covert channel (i.e. out-of-order transactions). Sorting by timestamp may be an essential component of some simulations or protocols.

Usage: `sort.py <key1> [<key2>] ... <file1> [<file2>] ...`

```
Valid sort keys:
-t or --timestamp
-a or --amount
-i or --id or --transactionid
```

```
--sb or --source-bsb
--db or --dest-bsb
--sa or --source-acct
--da or --dest-acct
Short-form options cannot be combined (e.g.  -tai is illegal)
```

The source code of the Sort program is in Appendix B.1.5 on page 60.

4.4.5 Filter

The filter program passes input transaction to output, including only transactions which match the specified criteria. The filter can be inverted (to exclude transactions matching the criteria).

This is one of the most useful operations for constructing protocols. For example, this program can be used to simulate a bank or branch removing transactions intended for itself from the network for processing, while passing transactions for other banks or branches onward.

Usage: `filter.py [options] <file1> [<file2>] ...`

```
-h, --help show this help message and exit
-v, --invert         Output non-matching records instead.

-t TIME, --timestamp=TIME
-a AMOUNT, --amount=AMOUNT
-i TID, --id=TID, --transactionid=TID
--sb=SOURCEBSB, --sourcebsb=SOURCEBSB
--db=DESTBSB, --destbsb=DESTBSB
--sa=SOURCEACCT, --sourceacct=SOURCEACCT
--da=DESTACCT, --destacct=DESTACCT
```

The source code of the Filter program is in Appendix B.1.6 on page 63.

4.4.6 Muxer

This component takes multiple CSV files and combines them with the rows in random order.

This can be used to combine transactions from several other fin in the simulator, and mix transactions from the INSERT program with "raw" transactions which have not been manipulated.

Usage: `muxer.py <module> <transactions.csv> [....csv]`

```
This program does not have any options.
```

The source code of the Muxer program is in Appendix B.1.4 on page 59.

4.5 Other Files

4.5.1 Transaction.py

The source code of the Transaction Generator is in Appendix B.1.1 on page 53. Most of the random generation code is in the Transaction class (see section 4.4.1.1 on page 32 and Appendix B.3.1 on page 71).

This file contains the Transaction class which does much of the underlying work of the above programs, including converting CSV strings to and from transaction objects and generating random transactions data as required.

This file also contains the developer notes on the appropriate code style and conventions for the project.

The transaction class currently handles only customer visible transaction data. Data which is not seen by either the sending or receiving account holders (e.g. clearing or routing data) is not stored in a transaction object.

The source code of the Transaction class is in Appendix B.3.1 on page 71.

4.5.2 Data Hiding Modules

The Data Hiding Modules each encapsulate a different data hiding algorithm The Insert and Extract programs require the name of the module used to be specified. The modules are each a single python file and each one is described in Section 5 on the next page.

4.5.3 Test Makefile

A simple GMAKE and BSDMAKE compatible makefile is included which tests each of the programs and modules by generating random transactions and performing all available operations on them. This file can be run on most Unix-like systems with the make command.

The contents of the Makefile is in Appendix B.3.2 on page 76.

5 Data Hiding Modules

The Insert and Extract programs require that a specific module be specified to perform information hiding. These modules encapsulate the methods which add the covert channel to the transaction data.

Some methods may only be usable by a bank, or by the initiator of a transaction. For example, methods which hide data in fields which only exist during the transaction would not be usable by a customer. Methods which require a certain amount to be transferred or require transactions to be sent at certain dates or times are more useful to customers.

5.1 Module Interface

The modules must implement three functions, described in this subsection.

As more complex data hiding algorithms are developed, it may be necessary for this interface to change.

5.1.1 InsertCode()

InsertCode() is called by the insert operation on each transaction which may be used to hide data. If required by the algorithm InsertCode maintains state between calls. It takes two arguments: the transaction to operate on, and the numeric code to insert. It returns a reference to the transaction.

5.1.2 ExtractCode()

ExtractCode() is called on each transaction to be analyzed for hidden data. ExtractCode maintains state between calls. It takes one argument (the transaction to analyze) and does not return anything.

5.1.3 GetCode()

GetCode() is used to retrieve the extracted code after the transactions have been analyzed. It returns a numeric code and does not take any arguments.

5.2 Amount

Similar to the example mentioned in the introduction (section 1 on page 8), this module stores the data in the cents of the amount transferred. The

second digit of the cents contains a digit of the code; the first digit contains the position of the data digit within the code. The name of the module implementing this algorithm is CC-AMOUNT.

No attempt is made by this algorithm to distinguish transactions which have hidden data and ones which do not. It is assumed that a second code (such as the timestamp method mentioned in the introduction scenario) will be used to distinguish transactions with hidden data.

By storing the position of the transferred digit, the algorithm is resilient to changes in the order of transactions. By default the position-digit cent values repeat, which makes this algorithm far easier for an adversary to detect, although it improves redundancy if not all transactions are received. To improve concealment, repeating could be turned off and/or the transactions mixed with transfers without hidden data (using a second code to distinguish these).

Although usable in practice this module is intended as a proof of concept and not for practical use. This algorithm could typically be used only by customers; banks could not "piggyback" a covert channel on their customers by changing the amounts that they were transferring.

The source code of cc-amount is in Appendix B.2.1 on page 66 and a sample simulation using this module is in Appendix A on page 50.

5.2.1 Example of CC-AMOUNT algorithm

Example code: 31459

Digit:	3	1	4	5	9
Position	4	3	2	1	0

Transaction Amounts:

- $??.**09**
- $??.**15**
- $??.**24**
- $??.**31**
- $??.**43**

5.3 Split-amount

This module (CC-SPLIT-AMT) takes each transaction and splits it into two; the cents figure in one of the pair determines the code in a manner similar to the amount module described in the previous subsection.

Before being cleared and funds transferred to the customer, the split transaction is joined to form the original.

The splitter relies on there being at least $1 in the transaction and enough transactions to convey the whole message (one per digit of code).

Like the CC-AMOUNT module described in the previous subsection, this is a proof of concept algorithm and not intended for practical use. Unlike the previous module, this one could not be used by a customer (as it requires manipulating the transaction ID), but could be used by banks to add a watermark to the transactions. As the sum of values of the split transactions are the same as that of the original transaction, and the original transaction values would appear on a customer statement, in most cases (barring the problems mentioned above) this technique should be transparent to a customer, although easily detectable by an observer FI within the financial network.

The ID of the new transactions must be related to the original ID in such a way that the original ID can be reformed. This is crudely performed by setting half the digits on one of the split pair to zero, on the other transaction, setting the other half to zero. This makes it very easy for an observer to detect the presence of these split transactions. This will also cause errors if a non-split transaction is created with a matching string of zeros in its ID.

Like the previous module, this technique is resilient to changes in the order of transactions as the location of each data digit is encoded in each message.

5.3.1 Example

Example code: 642

Digit:	6	4	2
Position	2	1	0

Original transaction amounts and IDs:

$90.00	72895721
$30.00	45312586
$60.00	75618676

Altered (split) transaction amounts and IDs:

Amount	ID	Amount	ID
\$29.**02**	72890000	\$60.98	00005721
\$14.**14**	45310000	\$15.86	00002586
\$43.**26**	75610000	\$16.74	00008676

5.4 Timestamp-LSD

This module (CC-TIME-LSD) stores the code in the least significant digit of the timestamp. This will produce a change of ± 9 seconds. Customers may not receive seconds on their statement (or even any time of day information, for some banks) so this method may be undetectable by the customers.

No attempt is made by this algorithm to distinguish transactions which have hidden data and ones which do not. It is assumed that a second code is used to distinguish transactions with hidden data if required.

Unlike the previous modules, the position data is not encoded in each message. This algorithm is therefore not resilient to missing transactions, or changes in the order of transactions.

The source code of cc-time-lsd is in Appendix B.2.2 on page 69 and a sample simulation using this module is in Appendix A on page 50.

5.4.1 Example

Example code: 963

Original timestamps:

- 183475032
- 183479141
- 183483257

Altered timestamps:

- 183475033
- 183479146
- 183483259

5.5 Unfinished Modules

Ideas and algorithms for several modules were developed but modules to implement them were not completed:

5.5.1 Split-ratio

Similar to split-amount, but the ratio of the amounts in the split transaction is used to encode the data.

5.5.2 Timestamp-Sion

This algorithm uses a data hiding technique developed by Sion [28] on the timestamp field.

5.5.3 Transaction-Order

This method reorders the transmission of records using a technique similar to that presented by Ramos [27].

6 Conclusion

6.1 Complications

There were three significant difficulties encountered working on this project:

The APCA documentation was long (approximately 200 pages per protocol), written using legal and financial terminology (unfamiliar to most computer scientists) and many sections were classified (not publicly available). The analysis of APCA protocols took up most of the early period of this project.

Literature searches for covert channels in finance were complicated as this is a new area of research, without many articles which were even remotely related. Financial networks and money laundering detection research is constantly evolving as well - some of the references used were released less than a month before this report.

Real transaction data - even anonymized transaction data - could not be obtained. Thus there is no real data to check the transaction generator and the output of other programs. Substantial work was involved in searching for this data, and in its absence, attempting to generate the most accurate random transactions possible.

6.2 Applications

There are two notable applications of this work:

Steganographic Analysis of Financial Transactions
As shown in scenario 1 (on page 8), covert channels can be used for communication by money laundering groups and other criminal organizations. Police and anti-financial-crime organizations[19] are interested in countermeasures, i.e. detecting, decoding and disrupting these messages.

This framework can be used for experiments to find countermeasures for the criminal communication system, by simulating a data hiding system and examining what can be seen at different points in the financial network.

Watermarking to Trace Large Scale, Small Value Money Laundering Schemes
This framework can also be used for experiments which use the data hiding mechanism as a watermark to detect money laundering networks. MLN which use many transactions of small value are difficult to detect using established techniques [20, 24]. Watermarking may be a solution to this problem.

Known suspects in the MLN can have watermarks applied to any transaction they make. Watermarks may be also applied to any transaction which a financial crime investigator or expert system has deemed "suspicious".

Accounts which receive watermarked transactions are noted. Within the system these accounts may also be given "suspicious" status and their transactions watermarked as well. The watermarks may encode a value which is decreased at each step, in order to determine the "degrees of separation" between suspicious accounts; in this way a graph is formed of the MLN.

6.3 Future Work

Steganalysis:

- Currently no software has been written to perform statistical analysis on the transaction data to determine the detectability of the covert channels.

Protocol Improvements:
There are many types of EFTN protocols, and the implemented protocol and record format is extremely simple when compared to protocols such as APCS and CECS (see section 2.2 on page 11). Many improvements can be made to the existing simulated protocols including:

- Clearing and settlement protocols are not implemented in detail yet.
- Special fields used for EFTPOS, credit card, and Cheque transactions
- Processing "Items" in transactions (e.g. for Cheque based transfers).
- Handling cancelled transactions and other parts of "higher" protocols.

Data Hiding Modules:

- Many methods for data hiding have not been implemented as modules. Some of these are discussed in section 5.5 on page 40.
- More flexibility in how the existing modules operate would be desirable. For example, the amount module is hardcoded to use 1 digit of position and one digit of data. The timestamp module uses the single least significant digit. These could be made variable so more digits can be used in each transaction.

Transaction Generator:

- There should be more flexibility in how transactions are generated - being able to specify maximum and minimum values on the command line should be implemented, along with different random distributions.
- Transaction time data - it is likely that the volume of transactions differs during different times of the day (business hours in particular) as well as on different days of the week, days of the month and periods of the year (e.g. approaching Christmas, taxation dates, end of financial year). The generator currently does not take this into account as we have no data on how it varies.

Node Management Tools:

- A tool to combine programs into protocols and nodes would be highly desirable. This tool would likely use a flowchart-like GUI to show the interconnected nature of the nodes.

7 Acknowledgments

- Simon Duff provided useful information regarding Australian banking networks.
- Vijaya Bhaskar, Tim Fry and Imad Moosa (all from the College of Business at RMIT University) provided feedback regarding sources for transaction data.

8 References

8.1 EFT Concepts and Models

[1] D. W. Davies and W. L. Price. *Security for computer networks: an introduction to data security in teleprocessing and electronic funds transfer (2nd ed.)*. John Wiley & Sons, Inc., New York, NY, USA, 1989.

8.2 Australian EFT Systems

[2] *CHEQUES ACT 1986*, 2007. Last amended 2007. Available from: http://www.austlii.edu.au/au/legis/cth/consol_act/ca198692/index.html [cited 2012.02.19].

[3] Commonwealth bank online support, March 2009. Available from: http://support.commbank.com.au/app/answers/detail/a_id/707/ [cited 2012.02.15].

[4] Australian Payments Clearing Association. *Annual Report 2011*, June 2011. Available from: http://ar2011.apca.com.au/ [cited 2012.02.19].

[5] Australian Payments Clearing Association Limited. *PROCEDURES for AUSTRALIAN PAPER CLEARING SYSTEM (CS1)*, December 1993. Last amended: 1 July 2010. Available from: http://www.apca.com.au/Public/apca01_live.nsf/WebPageDisplay/RP_APCS [cited 2011.12.09].

[6] Australian Payments Clearing Association Limited. *REGULATIONS for AUSTRALIAN PAPER CLEARING SYSTEM (CS1)*, December 1993. Last amended: 9 December 2009. Available from: http://www.apca.com.au/Public/apca01_live.nsf/WebPageDisplay/RP_APCS [cited 2011.12.21].

[7] Australian Payments Clearing Association Limited. *PROCEDURES for BULK ELECTRONIC CLEARING SYSTEM (CS2)*, December 1994. Available from: http://www.apca.com.au/docs/payment-systems/becs_procedures.pdf [cited 2011.12.09].

[8] Australian Payments Clearing Association Limited. *REGULATIONS for BULK ELECTRONIC CLEARING SYSTEM (CS2)*, December 1994. Available from: http://www.apca.com.au/Public/apca01_live.nsf/WebPageDisplay/RP_BECS [cited 2011.12.09].

[9] Australian Payments Clearing Association Limited. *PROCEDURES for HIGH VALUE CLEARING SYSTEM (CS4)*, August 1997. Available from: http://www.apca.com.au/docs/payment-systems/hvcs_procedures.pdf [cited 2011.12.09].

[10] Australian Payments Clearing Association Limited. *REGULATIONS for HIGH VALUE CLEARING SYSTEM (CS4)*, August 1997. Available from: http://www.apca.com.au/docs/payment-systems/hvcs_regulations.pdf [cited 2011.12.09].

[11] Australian Payments Clearing Association Limited. *CECS Manual for CONSUMER ELECTRONIC CLEARING SYSTEM (CS3)*, December 2000. Available from: http://www.apca.com.au/Public/apca01_live.nsf/WebPageDisplay/RP_CECS [cited 2011.12.09].

[12] Australian Payments Clearing Association Limited. *REGULATIONS for CONSUMER ELECTRONIC CLEARING SYSTEM (CS3)*, December 2000. Available from: http://www.apca.com.au/docs/payment-systems/cecs_regulations.pdf [cited 2011.12.09].

[13] Australian Payments Clearing Association Limited. *PROCEDURES for AUSTRALIAN CASH DISTRIBUTION AND EXCHANGE SYSTEM (CS5)*, December 2001. Available from: http://www.apca.com.au/docs/payment-systems/acdes_procedures.pdf [cited 2012.02.15].

[14] Australian Payments Clearing Association Limited. *REGULATIONS for AUSTRALIAN CASH DISTRIBUTION AND EXCHANGE SYSTEM (CS5)*, December 2001. Available from: http://www.apca.com.au/docs/payment-systems/acdes_regulations.pdf [cited 2012.02.15].

8.3 International EFT Systems

[15] Iso 15022, 1999. Available from: http://www.iso.org/iso/iso_catalogue/ [cited 2011.12.09].

[16] Iso 20022, 2004. Available from: http://www.iso.org/iso/iso_catalogue/ [cited 2011.12.09].

[17] Swift history, 2008. Available from: http://www.swift.com/about_swift/company_information/swift_history.page?lang=en [cited 2011.12.09].

[18] Swift in figures - fin traffic, September 2011. Available from: http://www.swift.com/about_swift/company_information/swift_in_figures/archive/index.page? [cited 2011.12.09].

8.4 Money Laundering Processes

[19] Jeanne K. Giraldo and Harold A. Trinkunas. *Terrorism financing and state responses: a comparative perspective*. Stanford University Press, 2007.

[20] Angela Samantha Maitland Irwin, Kim-Kwang Raymond Choo, and Lin Liu. An analysis of money laundering and terrorism financing typologies. *Journal of Money Laundering Control*, 15(1):85–111, December 2011. Available from: http://www.emeraldinsight.com/journals.htm?issn=1368-5201&volume=15&issue=1&articleid=17004748&show=abstract&PHPSESSID=fcqk9ufrtptip3u30qal4pe580 [cited 2012.02.15], doi:10.1108/13685201211194745.

8.5 Money Laundering Detection & Statistical Analysis

[21] Frank Benford. The law of anomalous numbers. *Proceedings of the American Philosophical Society*, 78(4):551–572, mar 1938.

[22] Theodore P. Hill. A statistical derivation of the significant-digit law. *Statistical Science*, 10:354–363, 1995. Available from: http://www.tphill.net/publications/BENFORD%20PAPERS/statisticalDerivationSigDigitLaw1995.pdf [cited 2011.12.09].

8.6 Data Mining Money Laundering Detection

[23] Zengan Gao and Mao Ye. A framework for data mining-based anti-money laundering research. *Journal of Money Laundering Control*, 10(2):170–179, May 2007. Available from: http://www.emeraldinsight.com/journals.htm?issn=1368-5201&volume=10&issue=2&articleid=1603663&show=html&PHPSESSID=1hsqa5srghgel8b6hsqgrq0gs4 [cited 2012.01.17], doi:10.1108/13685200710746875.

[24] Ted E Senator, Henry G Goldberg, Jerry Wooton, Matthew A Cottini, A. F. Umar Khan, Christina D Klinger, Winston M Llamas, Michael P Marrone, and Raphael W. H Wong. Financial crimes enforcement network AI system (FAIS) identifying potential money laundering from reports of large cash transactions. *AI Magazine*, 16(4):21, December 1995. Available from: http://www.aaai.org/ojs/index.php/aimagazine/article/view/1169 [cited 2012.01.17], doi:10.1609/aimag.v16i4.1169.

8.7 Covert Channels and Information Hiding

[25] Patrick Bajari and Jungwon Yeo. Auction design and tacit collusion in FCC spectrum auctions. *National Bureau of Economic Research Working Paper Series*, No. 14441, 2008. Available from: http://www.nber.org/papers/w14441 [cited 2012.02.15].

[26] Peter Cramton and Jesse Schwartz. Collusive bidding in the FCC spectrum auctions. Papers of Peter Cramton 02collude, University of Maryland, Department of Economics - Peter Cramton, 2002. Available from: http://ideas.repec.org/p/pcc/pccumd/02collude.html [cited 2012.02.15].

[27] Jorge R. Ramos. *Dynamic covert channels in finance*. PhD thesis, Purdue University, West Lafayette, IN, USA, 2007. AAI3287305.

8.8 Watermarking

[28] Radu Sion, Mikhail Atallah, and Sunil Prabhakar. On watermarking numeric sets. In *Proceedings of the Workshop on Digital Watermarking IWDW 2002*, Seoul, Korea, 2002. 2002, Lecture Notes in Computer Sciences, Springer Verlag (LNCS).

8.9 Miscellaneous

[29] Decimal fixed point and floating point arithmetic, python v2.7.2 documentation. http://docs.python.org/library/decimal.html. Available from: http://docs.python.org/library/decimal.html [cited 2012.02.15].

[30] Python programming language. http://www.python.org/, 1990. Available from: http://www.python.org/ [cited 2012.02.15].

[31] Iso/iec 9899:2011, 2011. Information technology – Programming languages – C. Available from: http://www.iso.org/iso/ [cited 2012.02.15].

[32] The IEEE and The Open Group. *Tee, The Open Group Base Specifications Issue 7, IEEE Std 1003.1-2008*, 2008. Available from: http://pubs.opengroup.org/onlinepubs/9699919799/utilities/tee.html [cited 2012.02.15].

Appendices

A Simulator Samples

This section contains several very simple examples of simulator usage. The operations are called directly, from the command line. In the intended usage, these programs would not be called directly, but by a tool which constructs financial network simulations of many protocols and nodes.

A.1 cc-amount

The following is a simple example of a code (47) hidden and extracted in a few transactions using the "Amount" algorithm:

```
> ./trans-gen.py 2 | tee /tmp/1.$$
Timestamp, Trans. ID, Amount, Src BSB, Src Acct, Dest Acct, Dest BSB
1326307380,143760206,16287.89,917654,34932956,79322,37059718
1327010633,443511746,11091.68,304417,417960,752228,74559508

> ./insert.py cc-amount 47 /tmp/1.$$ | tee /tmp/2.$$
1326307380,143760206,16287.07,917654,34932956,79322,37059718
1327010633,443511746,11091.14,304417,417960,752228,74559508

> ./extract.py cc-amount /tmp/2.$$
Code extracted: 47
```

A.2 cc-time-lsd

The following is a simple example of a code (47) hidden and extracted in a few transactions using the "Timetamp-LSD" algorithm:

```
> ./trans-gen.py 3 | tee /tmp/1.$$
Timestamp, Trans. ID, Amount, Src BSB, Src Acct, Dest Acct, Dest BSB
1326873450,108377283,3518.06,122021,14867629,18797,54860426
1325673276,204996482,3202.1,493422,55263511,245805,97876193
1327483531,633250232,1242.22,484894,27857799,7756,82752331
> ./insert.py cc-time-lsd
47 /tmp/1.$$ | tee /tmp/2.$$
```

```
1326873457,108377283,3518.06,122021,14867629,18797,54860426
1325673274,204996482,3202.1,493422,55263511,245805,97876193
1327483530,633250232,1242.22,484894,27857799,7756,82752331
> ./extract.py cc-time-lsd
/tmp/2.$$
Code extracted: 47
```

A.3 sort and filter

The following is a more involved example, showing several operations including sort and filter. The transactions are sorted by timestamp after generation. The value 107 is then inserted, but one transaction goes missing (filter operation) - this could be due to a transient failure within the FI. Due to the missing transaction the extract operation does not correctly recover the code.

```
>  ./trans-gen.py 5 | tee /tmp/1.$$
Timestamp, Trans. ID, Amount, Src BSB, Src Acct, Dest Acct, Dest BSB, Dest Name, Des
1327842663,782398631,13984.87,127709,85674744,30147,63238002,,
1326202531,815081374,5480.88,489410,6638234,468190,81508771,,
1326416078,86400640,762.11,653848,79783500,648441,55149951,,
1326749815,314197294,2762.07,268228,88157670,639852,37846209,,
1326136104,520214310,3631.82,711787,78480993,11526,56877919,,

./sort.py --timestamp /tmp/1.$$ | tee /tmp/2.$$
1326136104,520214310,3631.82,711787,78480993,11526,56877919,,
1326202531,815081374,5480.88,489410,6638234,468190,81508771,,
1326416078,86400640,762.11,653848,79783500,648441,55149951,,
1326749815,314197294,2762.07,268228,88157670,639852,37846209,,
1327842663,782398631,13984.87,127709,85674744,30147,63238002,,

./insert.py cc-time-lsd 107 /tmp/2.$$ | tee /tmp/3.$$
1326136107,520214310,3631.82,711787,78480993,11526,56877919,,
1326202530,815081374,5480.88,489410,6638234,468190,81508771,,
1326416071,86400640,762.11,653848,79783500,648441,55149951,,
1326749810,314197294,2762.07,268228,88157670,639852,37846209,,
1327842660,782398631,13984.87,127709,85674744,30147,63238002,,

./filter.py -v --db=468190 /tmp/3.$$ | tee /tmp/4.$$
1326136107,520214310,3631.82,711787,78480993,11526,56877919,,
```

```
1326416071,86400640,762.11,653848,79783500,648441,55149951,,
1326749810,314197294,2762.07,268228,88157670,639852,37846209,,
1327842660,782398631,13984.87,127709,85674744,30147,63238002,,

./extract.py cc-time-lsd /tmp/4.$$
Code extracted: 17
```

B Source Code

B.1 Operations

B.1.1 trans-gen.py

Listing 1: trans-gen.py

```
#!/usr/bin/env python
# $Id: trans-gen.py 238 2012-02-21 11:41:16Z e62840 $
#
#Transaction Generator
#
# NB: tee command can be used to print and redirect output.

'''The Transaction Generator outputs a CSV (comma separated value)
stream containing a random set of transactions.'''

# XXX: See transaction.py for code style/conventions.


import sys, transaction

#
    ============================================================

#                     Main
#
    ============================================================


# cli arguments check
if (len(sys.argv) < 2):
    sys.exit("Usage: trans-gen.py [options] <number>")

# TODO: If options are required, this sould be replaced when optparse
# is used.
try:
    numTrans = int(sys.argv[1])
except:
```

```
        sys.exit("Must give a number of transactions to print.
            ")

# TODO: possible future options
#  --seed - specify seed
#  - modify source acct, dest acct values
#  - modify trans ID values
#  - modify amount values
#      - modify distribution of amounts (power law -
    benfords law etc).
#  - specify timestamp format, start and end times

# print header
transaction.csvHeader()

# print each line
for i in range(numTrans):
    print transaction.randomTransaction()
```

B.1.2 insert.py

Listing 2: insert.py

```
#!/usr/bin/env python
# $Id: insert.py 142 2012-02-09 23:32:11Z e62840 $

'''This program hides data inside a set of transactions.
   Required
commandline arguments: the data hiding module (algorithm)
    to use, the
data to hide (an integer) and file(s) containing the
   transactions.
Standard input can be used by specifying "-" as the
   filename.'''

import fileinput, sys, warnings
from transaction import *

# XXX: See transaction.py for code style/conventions.

#
   ============================================================

#                         Main
#
   ============================================================


if (len(sys.argv) < 4):
    sys.exit("Usage: insert.py <module> <code> <
       transactions.csv> [....csv]")


# commandline arguments:

try:
    mod = __import__(sys.argv[1])       # run-time import
       function
except:
    sys.exit("Must give a covert channel module as the
       first argument. Don't include the .py extension.")

try:
```

```
30      code = int(sys.argv[2])
    except:
        sys.exit("Must give a positive integer code as the
            second argument.")

    # read all lines in all files
35
    for line in fileinput.input(sys.argv[3:]):
        tr = transFromStr(line)
        if (tr):
            print(mod.InsertCode(tr, code))
40          # XXX: note that InsertCode is resposible for
                keeping track of
            # its own state internally, if necessary.
        else:
            if (tr == False): # None == OK
                warnings.warn("Failed to parse transaction in {}
                    line {}".format(\
45                      fileinput.filename(), fileinput.
                            filelineno()))
    # end of file input parser
```

B.1.3 extract.py

Listing 3: extract.py

```
#!/usr/bin/env python
# $Id: extract.py 167 2012-02-14 07:20:08Z e62840 $

'''This program extracts hidden data from a set of
   transactions.
Required commandline arguments: the data hiding module (
   algorithm) to
use and file(s) containing the transactions.  Standard
   input can be
used by specifying "-" as the filename.'''

import fileinput, sys, warnings
from transaction import *

# XXX: See transaction.py for code style/conventions.

#
   ============================================================

#                        Main
#
   ============================================================


if (len(sys.argv) < 3):
   sys.exit("Usage: extract.py <module> <transactions.csv
      > [....csv]")


# commandline arguments:

try:
   mod = __import__(sys.argv[1])       # run-time import
      function
except:
   sys.exit("Must give a covert channel module as the
      first argument. Don't include the .py extension.")

# read all lines in all files
```

```
for line in fileinput.input(sys.argv[2:]):
    tr = transFromStr(line)
    if (tr):
        mod.ExtractCode(tr)
        # XXX: note that ExtractCode is resposible for
            keeping track of
        # its own state internally, if necessary.
    else:
        if (tr == False): # None == OK
            warnings.warn("Failed to parse transaction in
                {0} line {1}".format(\
                    fileinput.filename(), fileinput.
                        filelineno()))
# end of file input parser

print("Code extracted: {0}".format(mod.GetCode()))
```

B.1.4 muxer.py

Listing 4: muxer.py

```
#!/usr/bin/env python
# $Id: muxer.py 168 2012-02-14 09:47:22Z e62840 $

'''This component takes multiple transactions streams and
    combines
them with the rows in random order.'''

import fileinput, random, sys, warnings
from transaction import transFromStr

# XXX: See transaction.py for code style/conventions.

if (len(sys.argv) < 2):
    sys.exit("Usage: muxer.py <transactions.csv> [....csv]
       ")

# read all lines in all files
alllines = []
for line in fileinput.input(sys.argv[1:]):
    tr = transFromStr(line)
    if (tr):
        alllines.append(tr)
    else:
      if (tr == False): # None == OK
        warnings.warn("Failed to parse transaction in {0}
            line {1}".format(\
                fileinput.filename(), fileinput.
                    filelineno()))

# shuffle with random.shuffle, then print
random.shuffle(alllines)
for tr in alllines:
    print tr
```

B.1.5 sort.py

Listing 5: sort.py

```
#!/usr/bin/env python
# $Id: sort.py 281 2012-03-07 20:58:13Z e62840 $

"""Sorts the input transactions. The sort is stable and multiple
options can be specified (transactions are sorted repeatedly in
the order the arguments appear)"""

# XXX: See transaction.py for code style/conventions.

import fileinput, sys
from transaction import *

def quit_with_usage():
    '''Exits the program with a long usage message.'''
    sys.exit('''Usage: sort <key1> [<key2>] ... <file1> [<file2>] ...
    Valid sort keys:
        -t or --timestamp
        -a or --amount
        -i or --id or --transactionid
        --sb or --source-bsb
        --db or --dest-bsb
        --sa or --source-acct
        --da or --dest-acct
    Short-form options cannot be combined (e.g. -tai is illegal)
''')

# doing our own option parsing here, for two reasons:
#  (a) there aren't many options or any with arguments
#  (b) optparse doesn't guarantee returning the options in
#  order.
# (particularly b)

if (len(sys.argv) < 2):
    quit_with_usage()
```

```
# parse cli first (this allows bad args to error out before any
# processing is done). Construct a list of lambda functions to
# do the sort comparisons.
sortfuncs = []
filelist = []
for arg in sys.argv[1:]:
    if (arg == "-h" or arg == "--help"):
        quit_with_usage()

    # for each valid sort key...
    elif (arg == '-t' or arg == '--timestamp'):
        sortfuncs.append(lambda t: t.time_t)
    elif (arg == '-a' or arg == '--amount'):
        sortfuncs.append(lambda t: t.amount)
    elif (arg == '-i' or arg == '--id' or arg == '--
        transactionid'):
        sortfuncs.append(lambda t: t.trans_id)
    elif (arg == '--sb' or arg == '--source-bsb'):
        sortfuncs.append(lambda t: t.src_bsb)
    elif (arg == '--db' or arg == '--dest-bsb'):
        sortfuncs.append(lambda t: t.dst_bsb)
    elif (arg == '--sa' or arg == '--source-acct'):
        sortfuncs.append(lambda t: t.src_acct)
    elif (arg == '--da' or arg == '--dest-acct'):
        sortfuncs.append(lambda t: t.dst_acct)

    #if not an argument, try adding to file list...
    else:
        # TODO if starts with a - then warning?
        filelist.append(arg)

# read in all transactions in all files for sorting
alllines = []
for line in fileinput.input(filelist):
    tr = transFromStr(line)
    if (tr):
        alllines.append(tr)
    else:
      if (tr == False): # None == OK
         warnings.warn("Failed to parse transaction in {}
            line {}".format(\
```

```
                    fileinput.filename(), fileinput.
                        filelineno()))
76
    # sort transactions by all keys
    for sf in sortfuncs:
        alllines.sort(key=sf) # sf is a lambda function

81  # print final result
    for tr in alllines:
        print tr
```

B.1.6 filter.py

Listing 6: filter.py

```
#!/usr/bin/env python
# $Id: filter.py 239 2012-02-21 12:50:11Z e62840 $

import fileinput, sys, optparse, transaction
# NB: optparse deprecated since 3.2, but argparse needs 3.2+

# XXX: See transaction.py for code style/conventions.

def match(tr, options):
    '''Returns True if the transaction matches against ALL of the
specified options.'''

    # TODO: should be a way to do this with lambda functions...

    if (options.time is not None):
        if (tr.time_t != options.time):
            return False

    if (options.amount is not None):
        if (tr.amount != options.amount):
            return False

    if (options.tid is not None):
        if (tr.trans_id != options.tid):
            return False

    if (options.sourcebsb is not None):
        if (tr.src_bsb != options.sourcebsb):
            return False

    if (options.destbsb is not None):
        if (tr.dst_bsb != options.destbsb):
            return False

    if (options.sourceacct is not None):
        if (tr.src_acct != options.sourceacct):
            return False
```

```
        if (options.destacct is not None):
            if (tr.dst_acct != options.destacct):
                return False

        return True

    # main section begins here

    # option parsing
    parser = optparse.OptionParser()
    parser.add_option("-t", "--timestamp", dest="time")
    parser.add_option("-a", "--amount", dest="amount")
    parser.add_option("-i", "--id", "--transactionid", dest="
        tid")
    parser.add_option("--sb", "--sourcebsb", dest="sourcebsb"
        )
    parser.add_option("--db", "--destbsb", dest="destbsb")
    parser.add_option("--sa", "--sourceacct", dest="
        sourceacct")
    parser.add_option("--da", "--destacct", dest="destacct")

    # invert filter
    parser.add_option("-v", "--invert", action="store_true",
        dest="invert",
                      help="Output non-matching records
                          instead.")

    (options, args) = parser.parse_args()

    # TODO: range matches:
    # match on timstamp ><=
    # match on amount ><=

    if (len(args)==0):
        sys.exit("No inout files specified")

    # read all rows
    for line in fileinput.input(args):
        tr = transaction.transFromStr(line)
        if (tr):
            if (match(tr, options)):
                if (not options.invert):
```

```
                print tr
77          else:
               if (options.invert):
                  print tr
        else:
          if (tr == False): # None == OK
82           warnings.warn("Failed to parse transaction in {}
                line {}".format(\
                       fileinput.filename(), fileinput.
                          filelineno()))
```

B.2 Data Hiding Modules

B.2.1 cc-amount.py

Listing 7: cc-amount.py

```
# $Id: cc-amount.py 258 2012-02-22 18:51:01Z e62840 $

'''A covert channel module which changes the cents values to encode
data.  Unlike the other modules, this irrevocably changes the amount
value.  This module can thus not be used by "bank" nodes, but it is a
plausible way of communicating between "customer" nodes.
'''

import sys, decimal
from transaction import *

# XXX: See transaction.py for code style/conventions.

# The data is encoded as $xxxxx.pq where p is the position of the
# digit of the data; q is the digit itself.

def InsertCode(tr, code):

    # initial call?
    if (InsertCode.code == "init"):
        InsertCode.code = code
        if (code > 9999999999):
            sys.exit("Error: Code can be at most 10 digits long with this module.")

    # TODO: consider adding a new "InitInsert" function instead of this
    # workaround? (assuming we still check for initialisation here, it
    # does mean more code and function calls than the workaround), but
    # less argument passing

    # calc values for encoding
```

```
31      i = int(tr.amount)
        p = InsertCode.pos/10.0                    # .0 force
            float
        q = (InsertCode.code%10)/100.0

        # combine with str and round to get Decimal to use 2
            decimal places
36      tr.amount = decimal.Decimal(str(round(i+p+q, 2)))

        # state for next call

        InsertCode.pos += 1
41      InsertCode.code /= 10
        # wrap around pos/code for redundancy?
        if (InsertCode.code == 0):
            InsertCode.pos = 0
            InsertCode.code = code
46
        return tr

    # between-call state for InsertCode
    InsertCode.pos = 0
51  InsertCode.code = "init"

    def ExtractCode(tr):
        p = int((tr.amount*10)%10)
        q = int((tr.amount*100)%10)
56      ExtractCode.codeDict[p] = q

        # TODO: Create average of values recieved for
            redundancy, instead
        # of replacing digit in dictionary.

61      # debug print "position {} digit {}".format(p,q)

    # between-call state for ExtractCode and GetCode
    ExtractCode.codeDict = {}    # dictionary

66  def GetCode():
        code = 0
        for p in ExtractCode.codeDict:
            q =  ExtractCode.codeDict[p]
```

```
        # debug print "position {} digit {}".format(p,q)
71      code += (10**p) * q

    return code
```

B.2.2 cc-time-lsd.py

Listing 8: cc-time-lsd.py

```
# $Id: cc-time-lsd.py 281 2012-03-07 20:58:13Z e62840 $

'''A covert channel module which changes the least
   significant digit
of the timestamp to a digit of the code value. Not
   tolerant of
reordering of the transactions'''

import sys, decimal
from transaction import *

# XXX: See transaction.py for code style/conventions.

def InsertCode(tr, code):

    # initial call?
    if (InsertCode.code == "init"):
        InsertCode.code = code

    # TODO: consider adding a new "InitInsert" function
       instead of this
    # workaround? (assuming we still check for
       initialisation here, it
    # does mean more code and function calls than the
       workaround), but
    # less argument passing

    # calc values for encoding
    i = int(tr.time_t)
    old = i % 10
    new = InsertCode.code%10

    # rm old and insert new
    tr.time_t = i - old + new

    # state for next call
    InsertCode.code /= 10

    return tr
```

```
# between-call state for InsertCode
InsertCode.pos = 0
InsertCode.code = "init"

def ExtractCode(tr):
    ExtractCode.digits.append(int(tr.time_t)%10)

# between-call state for ExtractCode and GetCode
ExtractCode.digits = []

def GetCode():
    code = 0
    p = 0
    for d in ExtractCode.digits:
        code += (10**p) * d
        p += 1

    return code
```

B.3 Miscellaneous

B.3.1 transaction.py

Listing 9: transaction.py

```
# $Id: transaction.py 160 2012-02-12 14:24:01Z e62840 $

# Class for a single transaction, and functions to manipulate them.

#============================================================

# Style and Conventions:
#
#============================================================

#
# Things that need to be fixed now (e.g. bugfixes) are marked FIXME
# Features to be implemented at a later date are marked TODO
# Things that developers need to watch out for are marked XXX
#
# Constants in UPPERCASE.
# Class definitions in PascalCase.
# Global scope objects (including functions) in camelCase.
# Local variables in lowercase.
#
# Sections seperated by headers using ==== lines.
# Sections: Constants, Classes, ____ Functions, Main
#           Constants include option defaults and precompiled regexes.

import re, decimal, random

#============================================================

# Constants
```

```
#
    ============================================================

27
MINTIME = 1325336400 # = 2012-01-01 00:00:00
MAXTIME = 1328014800 # = 2012-02-01 00:00:00

MAXID = 999999999    # 9 digits
32 MAXACCT = 99999999    # 8 digits
MAXBSB = 999999    # 8 digits

MINAMOUNT = 1
MAXAMOUNT = 1000000     # 1 million
37 #AMOUNTMODE = 250
AMOUNTMU = 8.0257
AMOUNTSIGMA = 1

CSVHEADER = "Timestamp, Trans. ID, Amount, Src BSB, Src
    Acct, Dest Acct, Dest BSB, Dest Name, Description"
42 TRANS_CSV_RE = re.compile(r'^(\d+),(\d+),([0-9.]+),(\d+)
    ,(\d+),(\d+),(\d+),([ \w]*),([ \w]*)$')
# regex matches header above - note strings are optional
    and may
# be empty.

#
    ============================================================

47 # Classes
#
    ============================================================


class Transaction(object):
    '''Encapsulates a single wire transfer transaction
        from one account
52      to another. Only customer-facing data is stored -
            any routing
        information is NOT part of this class.'''

    def __init__(self, time_t, trans_id, amount, src_bsb,
        \
```

```
                src_acct, dst_bsb, dst_acct, dst_name="",
                    desc=""):
57          # TODO validation!
            self.time_t = time_t
            self.trans_id = trans_id
            self.amount = decimal.Decimal(amount)
            self.src_bsb = src_bsb
62          self.src_acct = src_acct
            self.dst_bsb = dst_bsb
            self.dst_acct = dst_acct
            self.dst_name = dst_name
            self.desc = desc
67
        def match_tsd(self, trans):
            '''Returns true if the transaction passed has the
               same
               timestamp, source and destination as this one.
                  '''
            if (self.time_t == trans.time_t and \
72              self.src_bsb == trans.src_bsb and \
                self.src_acct == trans.src_acct and \
                self.dst_bsb == trans.dst_bsb and \
                self.dst_acct == trans.dst_acct):
                  return True
77
            return False

        def __str__(self):
            return "%s,%s,%s,%s,%s,%s,%s,%s,%s"%(self.time_t,\
82                self.trans_id,self.amount,\
                  self.src_bsb,self.src_acct,\
                  self.dst_bsb,self.dst_acct,self.dst_name,\
                  self.desc)


87
    # end of Transaction class

    #
        ============================================================

    # Related Functions
92  #
        ============================================================
```

```
def genRandomAmount():
    '''Returns a randomly generated transaction amount.
       This amount
follows an appropriate PDF and will not return values
    outside the
specified maximum or minimum for amount.'''

    amt = random.lognormvariate(AMOUNTMU, AMOUNTSIGMA)
    # round to 2 decimal places for cents
    amt = round(amt,2)

    if (amt > MAXAMOUNT):
        return genRandomAmount()
    if (amt < MINAMOUNT):
        return genRandomAmount()

    return amt

# TODO: add option to use the old triangular distribution
    ?
#       str(round(random.triangular(MINAMOUNT,MAXAMOUNT,
    AMOUNTMODE),2)), \

def randomTransaction():
    '''Returns a randomly generated transaction.'''
    return Transaction( \
        random.randint(MINTIME,MAXTIME), \
        random.randint(1,MAXID), \
        str(genRandomAmount()), \
        random.randint(1,MAXBSB), \
        random.randint(1,MAXACCT), \
        random.randint(1,MAXBSB), \
        random.randint(1,MAXACCT))

def transFromStr(st):
    '''Given a CSV string, returns a Transaction object,
       or False if
       the string is not a valid Transaction, or None if
          the string is
       blank or a header line.'''
```

```
    mat = TRANS_CSV_RE.match(st)
    if (mat):
        try:
            # TODO: something about the magic numbers here
                ...
            return Transaction(mat.group(1),mat.group(2),\
                    mat.group(3),mat.group(4),mat.group(5),
                        \
                    mat.group(6),mat.group(7),mat.group(8),
                       mat.group(9))
        except:
            return False
    else:
        if (st == ""):
            return None
        if (st.startswith(CSVHEADER)): # startswith allows
            extensions to format
            return None
        return False

def csvHeader():
    '''Prints the CSV header line.'''
    print CSVHEADER
```

B.3.2 Makefile

Listing 10: Makefile

```
#
# $Id: Makefile 281 2012-03-07 20:58:13Z e62840 $
#
# Test suite for financial transaction simulator
#

# Non-pipe tests will actually generate 2x these transations, for muxing.
# This is 1/10 of the average amount of direct credit transactions a
# Tier 1 BECS member processes each day.
NUMTRANS=3615
PIPETRANS=7230

# for goanna.cs.rmit
PY="python2.6"

################################################

# Meta-targets

default: test-file

test-file: test-gen test-mux test-sort test-filter test-amount test-timelsd

test-both: test-file test-pipe

################################################

# Targets using intermediate files

test-gen:
	${PY} trans-gen.py ${NUMTRANS} > /tmp/testsim-$$-1.csv
	${PY} trans-gen.py ${NUMTRANS} > /tmp/testsim-$$-2.csv

test-mux:
```

```
	${PY} muxer.py /tmp/testsim-$$-1.csv /tmp/testsim-$$
	    -2.csv > /tmp/testsim-$$-3.csv

test-sort:
	${PY} sort.py -t /tmp/testsim-$$-3.csv > /tmp/
	    testsim-$$-4.csv

test-filter:
	${PY} filter.py -v -t 10 /tmp/testsim-$$-4.csv > /
	    tmp/testsim-$$-5.csv

################################################

# Module targets using intermediate files

test-amount:
	${PY} insert.py cc-amount 987654321 /tmp/testsim-$$
	    -5.csv > /tmp/testsim-$$-6.csv
	${PY} extract.py cc-amount /tmp/testsim-$$-6.csv

test-timelsd:
	${PY} insert.py cc-time-lsd 12345678987654321 /tmp/
	    testsim-$$-5.csv > /tmp/testsim-$$-7.csv
	${PY} extract.py cc-time-lsd /tmp/testsim-$$-7.csv

# TODO: sorted test for consistency after splits

################################################

test-pipe:
	${PY} trans-gen.py ${PIPETRANS} | ${PY} muxer.py - |
	     ${PY} sort.py -t - | \
	    ${PY} filter.py -v -t 10 - | ${PY} insert.py cc-
	       amount 987654321 - | \
	    ${PY} extract.py cc-amount -
	${PY} trans-gen.py ${PIPETRANS} | ${PY} muxer.py - |
	     ${PY} sort.py -t - | \
	    ${PY} filter.py -v -t 10 - | ${PY} insert.py cc-
	       time-lsd 12345678987654321 - \
	    | ${PY} extract.py cc-time-lsd -
```

C Uncited References

The following is a partial list of documents which were read (at least in part) during this project, entered in the bibliographic database, but not cited in the report.

C.1 EFT Concepts and Models

[33] Burton Rosenberg. *Handbook of Financial Cryptography and Security*. CRC Press, February 2010.

C.2 Australian EFT Systems

(No uncited references entered)

C.3 International EFT Systems

[34] J. Gustin and A. Goyens. Rfc 3615 - a uniform resource name (urn) namespace for swift financial messaging, September 2003. Available from: http://www.faqs.org/rfcs/rfc3615.html [cited 2011.12.09].

C.4 Money Laundering Processes

(No uncited references entered)

C.5 Money Laundering Detection & Statistical Analysis

[35] Journal of forensic accounting: Recent contents. http://www.rtedwards.com/journals/JFA/contents2.html. Available from: http://www.rtedwards.com/journals/JFA/contents2.html [cited 2011.12.09].

[36] Hsinchun Chen, Fei-Yue Wang, Christopher C. Yang, Daniel Dajun Zeng, Michael Chau, and Kuiyu Chang, editors. *Intelligence and Security Informatics, International Workshop, WISI 2006, Singapore, April 9, 2006, Proceedings*, volume 3917 of *Lecture Notes in Computer Science*. Springer, 2006.

[37] Christian Kleiber and Samuel Kotz. *Statistical Size Distributions in Economics and Actuarial Sciences*. John Wiley and Sons, 2003.

[38] Simon Newcomb. Note on the frequency of use of the different digits in natural numbers. *American Journal of Mathematics*, 4(1/4):39–40, 1881. doi:10.2307/2369148.

[39] M. E. J. Newman. Power laws, pareto distributions and zipfs law. *Contemporary Physics*, 46(5):323–351, 2005. doi:10.1080/00107510500052444.

[40] Mark J. Nigrini. I've got your number. *Journal of Accountancy*, may 1999. Available from: http://www.journalofaccountancy.com/Issues/1999/May/nigrini [cited 2011.12.09].

[41] Huizhang Shen, Jidi Zhao, and Huanchen Wang. Illegal intrusion detection based on hidden information database. In *WISI*, pages 79–84, 2006.

[42] Hal Varian. Benford's law. *The American Statistician*, 26:65.

C.6 Data Mining Money Laundering Detection

[43] E. L Barse, H. Kvarnstrom, and E. Jonsson. Synthesizing test data for fraud detection systems. In *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, pages 384– 394. IEEE, December 2003. Available from: http://ieeexplore.ieee.org.ezproxy.lib.rmit.edu.au/xpls/abs_all.jsp?arnumber=1254343&tag=1 [cited 2012.01.17], doi:10.1109/CSAC.2003.1254343.

[44] Nhien An Le Khac and M. Kechadi. Application of data mining for anti-money laundering detection: A case study. In *2010 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 577–584. IEEE, December 2010. Available from: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?reload=true&arnumber=5693349 [cited 2012.01.17], doi:10.1109/ICDMW.2010.66.

[45] R. CORY Watkins, K. Michael Reynolds, Ron Demara, Michael Georgiopoulos, Avelino Gonzalez, and Ron Eaglin. Tracking dirty proceeds: Exploring data mining technologies as tools to investigate money laundering. *Police Practice and Research*, 4(2):163–178, 2003. Available from: http://www.tandfonline.com.ezproxy.lib.

rmit.edu.au/doi/abs/10.1080/15614260308020 [cited 2012.01.17], doi:10.1080/15614260308020.

[46] Jennifer J. Xu and Hsinchun Chen. Fighting organized crimes: using shortest-path algorithms to identify associations in criminal networks. *Decision Support Systems*, 38(3):473–487, December 2004. Available from: http://www.sciencedirect.com/science/article/pii/S0167923603001179 [cited 2012.01.17], doi:10.1016/S0167-9236(03)00117-9.

C.7 Covert Channels and Information Hiding

[47] Fabien A. P. Petitcolas, Ross J. Anderson, and Markus G. Kuhn. Information hiding – a survey, 1999.

C.8 Watermarking

[48] Brian Chen and Gregory W Wornell. Quantization index modulation: A class of provably good methods for digital watermarking and information embedding. *IEEE TRANS. ON INFORMATION THEORY*, 47:1423—1443, 1999. Available from: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.9769 [cited 2011.12.08].

[49] Ingemar J. Cox, Matthew Miller, and Jeffrey Bloom. *Digital watermarking*. Morgan Kaufmann, 2002.

C.9 Miscellaneous

[50] The c standard; incorporating technical corrigendum 1; BS ISO/IEC 9899:1999. *Scitech Book News*, 28(2):n/a, June 2004.