



9-30-2013


## Money Laundering Detection Framework to Link the Disparate and Evolving Schemes

Murad Mehmet  
*George Mason University*

Duminda Wijesekera  
*George Mason University*

Miguel F. Buchholtz  
*George Mason University*

Follow this and additional works at: <https://commons.erau.edu/jdfsl>

 Part of the [Computer Engineering Commons](#), [Computer Law Commons](#), [Electrical and Computer Engineering Commons](#), [Forensic Science and Technology Commons](#), and the [Information Security Commons](#)

### Recommended Citation

Mehmet, Murad; Wijesekera, Duminda; and Buchholtz, Miguel F. (2013) "Money Laundering Detection Framework to Link the Disparate and Evolving Schemes," *Journal of Digital Forensics, Security and Law*. Vol. 8 : No. 3 , Article 3.

DOI: <https://doi.org/10.15394/jdfsl.2013.1150>

Available at: <https://commons.erau.edu/jdfsl/vol8/iss3/3>

This Article is brought to you for free and open access by the Journals at Scholarly Commons. It has been accepted for inclusion in Journal of Digital Forensics, Security and Law by an authorized administrator of Scholarly Commons. For more information, please contact [commons@erau.edu](mailto:commons@erau.edu).



# **MONEY LAUNDERING DETECTION FRAMEWORK TO LINK THE DISPARATE AND EVOLVING SCHEMES**

Murad Mehmet  
Duminda Wijesekera  
[dwijesek@gmu.edu](mailto:dwijesek@gmu.edu)

and

Miguel Fuentes Buchholtz  
Department of Computer Science  
Volgenau School of Engineering  
George Mason University  
Fairfax, VA

## **ABSTRACT**

Money launderers hide traces of their transactions with the involvement of entities that participate in sophisticated schemes. Money laundering detection requires unraveling concealed connections among multiple but seemingly unrelated human money laundering networks, ties among actors of those schemes, and amounts of funds transferred among those entities. The link among small networks, either financial or social, is the primary factor that facilitates money laundering. Hence, the analysis of relations among money laundering networks is required to present the full structure of complex schemes. We propose a framework that uses sequence matching, case-based analysis, social network analysis, and complex event processing to detect money laundering. Our framework captures an ongoing single scheme as an event, and associations among such ongoing sequence of events to capture complex relationships among evolving money laundering schemes. The framework can detect associated multiple money laundering networks even in the absence of some evidence. We validated the accuracy of detecting evolving money laundering schemes using a multi-phases test methodology. Our test used data generated from real-life cases, and extrapolated to generate more data from real-life schemes generator that we implemented.

**Keywords:** Anti Money Laundering, Social Network Analysis, Complex Event Processing

## **1. INTRODUCTION**

Current Anti Money Laundering (AML) systems are designed to function based on the requirements of adopting organization. They vary from the multi-component and complex systems such as FINCEN (FAIS) to the specialized

single-purpose systems used by banks to report Due Diligence and Suspicious Activity Reports (SAR). To capture increasingly complex money laundering schemes (MLS) call for integrating new techniques such as Social Network Analysis (SNA) (Wasserman et al., 1994), in addition to the already used rule based analysis and risk modeling. An efficient AML system must have many components, where some components are being purely deterministic and others being purely probabilistic. An example of a probabilistic component is the risk analysis, and SNA is an example of a deterministic component. Generally, deterministic models consider social aspects and statistical models consider financial aspects (Wasserman et al., 1994).

The FINCEN AI System (FAIS) (Senator et al., 1995, 1996, 1998) designed for internal use analyzes SARs filed by banks. The system combines offline SAR data analyzed by human experts to identify possible hidden linkage among transactions using link analysis techniques. However, FAIS (Senator et al., 1995, 1996, 1998) only links and evaluates the database (DB) of the reported suspicious transactions offline. KDPprevent (Jacobs et al., 2003; Kuns et al., 2004) by KDLabs, a commercial product/service utilized by banks in Switzerland collect customer, account and transaction information for offline analysis, combining data-mining techniques with expert legal knowledge of legal experts.

Two models of sequence matching and link analysis (Liu et al., 2008; Schwartz et al., 2008) are relevant to our research in detecting evolving patterns of sequence. Liu (Liu et al., 2008) proposes a sequence matching algorithm to discover suspicious transaction sequences, using transaction histories of an individual's accounts and transaction information histories from a peer group. (Liu et al., 2008) focus only on the bank transactions, without covering other financial transactions such as stock market. Schwartz (Schwartz et al., 2008) proposes a model to find criminal networks using social network analysis, building upon Borgatti's SNA-based key player approach (Schwartz et al., 2008). One drawback of Borgatti's model is the failure to assign weights to actors and actor-actor relationships. Gunestas, et al.'s framework (Gunestas et al., 2008, 2010) is similar to ours, but with a narrower focuses on detecting Ponzi schemes.

The rest of the paper is organized as follows: Section 2 explains the Money Laundering Evolution Detection Framework (MLEDF) and Section 3 describe proposes a new ML detection algorithm. Section 4 evaluates the performance results of MLEDF using real-life cases. Section 5 describes related work and Section 6 concludes the paper.

## **2. MONEY LAUNDERING EVOLUTION DETECTION FRAMEWORK (MLEDF)**

The framework is composed of four different phases. Each phase will communicate with the next phase, and the output generated from each phase is sent into the next phase. The phases and their function are explained below.

*Transaction Data Collection:* The transaction agents or data input collectors from Automated Clearing House such as (EPN, FEDWIRE, and CHIPS) will send in their data format. The different types of transaction data are: Banking, Stock market, Derivative market, Web Services, Trading, Electronic Money, and Money Brokering. Once the industry-specific transaction data is gathered, relevant information is extracted for analysis. For example, there are more than 20 fields in stock order forms and we use only time, sender, receiver, price, quantity, symbol, market, sellerOrderID, buyerOrderID, tradeID, and country. Also, we use transaction-independent data used in the analysis, such as the economic status of the country, sales trends of the stock, and the stock value during the day.

1. *Data Processing:* The data collected from different systems are used to create patterns of the well-known MLS (Mehmet et al., 2010). The MLS-related data that is extracted from the streaming events is filtered before submitting them into the detection algorithms. The extracted data associated with each MLS pattern assigned to a specific MLS type using the following components:

- a) **Business Rules:** MLS business rules and red flags associated with each pattern, the rules associated with specific sector are used by the MLS detection algorithms to identify the MLS patterns.
- b) **MLS Template:** Well-known MLS templates will be used during this phase. Currently, the templates have seven major pattern types with their different subtype combinations. This acts as a repository of known MLS. If a new form of MLS is discovered, then it will be added to this DB.
- c) **ML Economic Models:** Three ML economic models (Mehmet et al., 2013) will be used to validate and increase the accuracy of the detection algorithms for well-known ML patterns. Those economic models determine if the evolution of MLS is within the accepted trend of the models.

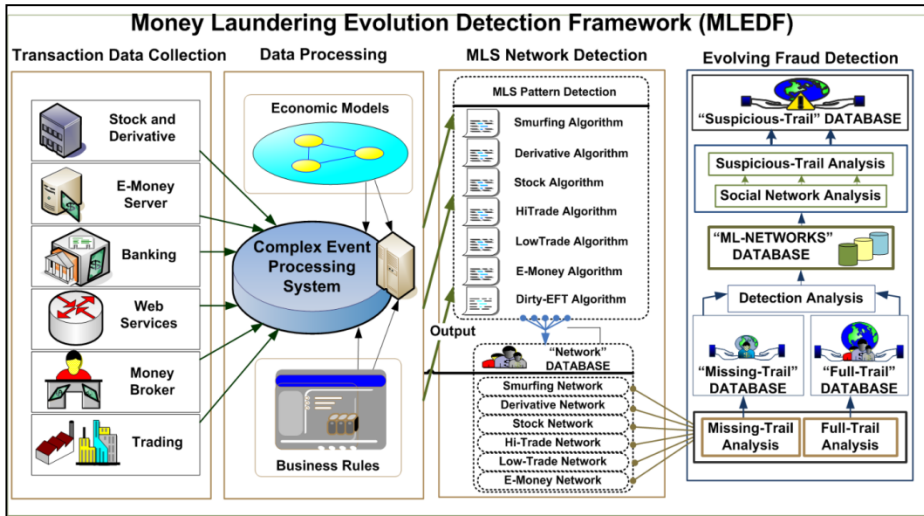


Figure 1 The MLEDF framework

2. *MLS Algorithms and MLS Network Detection:* There are six major heterogeneous algorithm modules (Smurfing, Trade, Stock, Derivative, EMoney, DirtyEFT). Each algorithm uses a different method to capture the network associated with the specific type of MLS. In real-time, the algorithms output, the discovered networks associated with the specific MLS patterns, each into a different database. Then, the discovered networks are reformatted and saved in a single database referred to as the “Network” Database. This process facilitates faster and efficient analysis of the links among MLS networks.
  
3. *Evolution Detection Analysis and Generating the Fraud Trail and Suspicious Trail:* Four separate algorithms are run to find the “Full-Trail”, “Missing-Trail”, and “Suspicious-Trail” (Mehmet et al., 2013) of MLS networks, and saved in separate databases. Full-Trail is a long series of MLS’s that span over many countries and involves many cycles of MLS. In essence, it is a concatenated sequence of related schemes (MLS) act in itself to transfer money from one MLS to the other until it reaches the final MLS, where we refer to the orchestrator (i.e., the money launderer) as the “EndBoss” in the final MLS. Any MLS or trail will have the originator “StartBoss” and the terminator “EndBoss”, in addition to the associates that maintain the MLS or trail. The “Associates” are the list of the people involved in the sequence of detected fraud. The “StartBoss” is the entity whom starts the MLS or trail. Missing-Trail is a short Full-Trail that does not exceed the depths of three related MLSs. We assume that the Missing-Trail is a premature Full-Trail with broken parts and missing links or evidence. A Suspicious-Trail is a combination of discovered Full-Trails and/or Missing-Trails, it will be constructed using algorithms that

incorporate SNA and numerical analysis techniques. The module “Detection Analysis” (Mehmet et al., 2013) determines the evolution of the “Full-Trail”’s such as the change to the number of involved associates, the changes to the cost of laundering, and changes to the laundering locations.

### **2.1 MLS Pattern Detection Modules in MLEDF**

MLEDF process is fed by data from many types of transaction data, where each feed is from a particular market or a finance industry. The main detection module is divided into sub-modules, where each sub-module detects money laundering patterns relevant to that specific market. This is because the data set of each market and industry is different than others and the money laundering techniques differ between them. MLEDF uses a core set of business rules to detect the evolution of MLS reported annually by FATF, with six detection patterns for each finance sector that we chose to include in our experiment.

### **2.2 Gathering Transaction Data and Generation of the Transaction Evidence Data**

A “Message” sent between two parties in the framework consists of the following components: (1) Common Mandatory Fields: Sender, Receiver, Time, Transaction ID, and a field that reflects the amount of funds transferred or price of the transaction; (2) Pattern Specific Mandatory Fields: A set of attributes pertinent to the transaction type. For example, the Smurfing transaction will have only the "EFT" field that reflects whether the banking transaction is an EFT or not. The stock transaction has more fields as in "Quantity, Symbol, Market, TradeID, Country, etc."; and (3) Auxiliary Third Party Fields: The framework retrieves critical data from third party sources, used in conjunction with the transaction data. The auxiliary data includes information such as recent market stock and derivative data, current product market price, and country economic status. This data is used to compare the transaction price and product price with the nominal price.

The “Comprehensive Output” is the MLS pattern-specific output generated by the MLS detection algorithm. As seen in figure 2, the output produces comprehensive evidence for each detected pattern, and it is different from other patterns. The output is saved in a separate database. For example, the field “Associates” exists in most of the outputs, it is a list of the people involved in the sequence of detected fraud. The size of the list varies because the list will expand as the money transfers from one entity to the next entity, until it reaches the final destination or terminates with a fund withdrawal.

The number of the transactions (which can be modeled as steps in an event) required to create a MLS vary based on the MLS type. We link the set of transactions that constitute the comprehensive-output for a specific MLS. Any previously examined transaction that is related to the current transaction under

examination is linked to the output of the current transaction, if the transactions share common fields and involve a fund transfer. For example, all the associates who are involved in a Smurfing fraud will be placed in the list of “Associates”, and the manipulator is represented in the field “Boss”.

All the data gathered from pattern detection are reduced to the minimum set that represents what we define as the “Network”, which constitutes the entities created the MLS and information about MLS. These entities are “EndBoss”, “StartBoss”, and “Associates”. The “Network” DB contains fields of participants and vital information of a detected MLS. Each network will be assigned a unique ID. The nine fields in the “Network” are: NetworkID, EndBoss, StartBoss, AmountLaundered, Associates, Type, DetectionTime, StartTime, EndTime. A network can be associated only with one type of MLS; therefore, the field “Types” represents the various well-known types of MLS (Mehmet et al., 2010).

The evolution-output “Detection-Schema” is generated by the “Detection-Analysis” module. The DB “Detection-Schema” contains information about the evolution of the ML trail, such as information of average cost and average number of associates used in each of the sequence of MLSs in the Full-Trail.

<b>Output High/Low Invoicing</b> 1- Time 2- Sender 3- Receiver 4- ProductID 5- Quantity 6- Location 7- MarketPrice 8- TotalLaundered 9- PaidPerUnit 10- EndBoss 11- StartBoss 12- StartTime 13- EndTime	<b>Output Stock</b> 1- Sender 2- Symbol 3- Time 4- Receiver 5- SellerExecutingFirm 6- BuyerExecutingFirm 7- Price 8- Quantity 9- Market 10- SellerOrderID 11- BuyerOrderID 12- TradeID 13- SellerOrderRemaining 14- BuyerOrderRemaining 15- Country 16- Day 17- IsCountryGoodRated 18- IsCompanyGoodRated 19- Last3DaysExpAvg (List) 20- Status 21- AssociateTransactions (List) 22- DistinctStatuses 23- FirstStatus 24- AssociatesAmountSum 25- Associates (List) 26- StartTime 27- EndTime 28- AmountLaundered 29- EndBoss 30- StartBoss
<b>Output Smurf/ Hawala/ Emoney</b> 1- LastReceiver 2- TransactionCount 3- FirstAmount 4- AllNodeLists (List) 5- FinalAmount 6- TotalCost 7- Associates (List) 8- StartTime 9- EndTime 10- Location 11- Type 12- NetworkID	<b>Output Derivative</b> 1- Buyer 2- OptionID 3- Symbol 4- Market 5- Call 6- SellerClientID 7- BuyerClientID 8- Quantity 9- StrikePnceHasProvided 10- Price 11- maturity 12- StrikeDate 13- SpotPrice 14- Exercise_optionID 15- Exercise_spotPrice 16- Exercise_exercise 17- Exercise_position 18- Exercise_lossGain 19- Location 20- NoLosses 21- PossibleAssociates (List) 22- PossibleAssociatesSum 23- PossibleAssociateIDs 24- EndBoss 25- StartBoss 26- StartTime 27- EndTime
	<b>Output DirtyEFT</b> 1- EndBoss 2- StartBoss 3- Associates (List) 4- Amount 5- NetworkID 6- DirtyEFTID 7- StartTime 8- EndTime 9- DetectionTime 10- WithdrawalTime 11- Location 12- Type 13- SubType 14- Cost 15- UsedUpAmount

Figure 2 The “Comprehensive Output” (DB content) of the Output of Six MLS Detection Algorithms

### 3. SOCIAL NETWORK ANALYSIS MODULE TO CREATE A “SUSPICIOUS-TRAIL”

There are many cases in which money launderers intentionally obfuscate the money laundering trail, either by hiding it (for instance by increasing the

transaction quantity and reducing the transaction amount), or performing it in a none-reported method as in Hawala. It becomes the task of an AML to detect these concealed relations and transactions. As a solution, we offer an additional module for social network analysis among transactions to unravel the hidden relations among MLS networks. MLEDF is designed from bottom-up with the concept of detecting and linking MLS trails (networks) even with missing evidence.

The major task of this module is to detect components of an actual “Full-Trail” even if there is a missing piece of evidence. The module will investigate the available trails (Full-Trail and Missing-Trail) by using our SNA DB that contains the weights of relationships among MLS participants. This is in order to determine if two trails are related by considering some attributes such as the amount of funds involved, location, affinity of participants, time, and methods used for laundering.

The SNA module is more resource consuming when compared to other modules, due to the extensive use of SNA, and link and weight calculations. The “Suspicious-Trail” module uses the “SNA” module to produce a new trail. This new trail contains two or more trails that are related based on SNA, even if we do not have captured a transaction joining them or any other evidence. The new trail is created after making a scientific calculation based on (SNA) results of a possible relationship between two or more “Full-Trails” and “Missing-Trails”.

The generated evolution patterns and strategies are collected into the “Suspicious-Trail” Database. This module contains the “SNA” sub-module that calculates and assesses the social network connections of individuals, peer-groups, and money laundering trails. The sub-module “SNA” is used to derive the associated suspicious trails based on the techniques of SNA. The table “Weight” that is used within the module is completely constructed with data output from running the “SNA” module.

### **3.1 Using Complex Event Processing in the Social Network Analysis Module**

Any MLS has the originator “StartBoss” and the terminator “EndBoss”, in addition to the associates that maintains the MLS. The output of MLS detection algorithms contains information about the participants, in addition to information such as amount laundered, final amount of funds, type of MLS, duration and start and end time. The critical question that ML experts contend to answer is “How fast and how well can we relate the different events in this universe of detected MLS?” Using the introduction of Complex Event Processing (CEP) systems like StreamBase, we developed an algorithm to create the full and accurate chains of related MLSs, such relations are used to transfer a fund to the next MLS until it reaches the final destination. That is,



the flow of the dirty money never stops until it reached the ultimate account. This cycle continues until it reached the final destination where the money launderer withdraws the money. Current AML systems have scalability issues in associating the multitude of different events of various MLS. We model each detected MLS as an event, and have various patterns of events categorized under six different types of MLS. For example, Full-Trail algorithm outputs a trail by using the functionality of CEP of perceiving the MLSs as a set of events. Without the CEP the MLS should dissolve into the constituent transactions to be analyzed and linked with the other transactions from another MLS (Time consuming and resource consuming). The CEP can link MLSs, perceived as events, using various criteria without the need to add more complex sub-algorithms for each criterion. That is, the Full-Trail connects the dots that exist, but it is harder and slower to connect them without CEP capabilities. Full-Trail captures the trail in cases where all evidence is available, whereas the Suspicious-Trail attempts to construct the path where some edges along the path is missing.

### **3.2 Integrating the “SNA” Module into MLEDF**

The major objective of the “SNA” module is to detect components of a undiscovered Full-Trails by performing analysis on the four databases “Network”, “Detection-Schema”, “Full-Trail”, and “Missing-Trail”. During the relationship analysis stage, the SNA module investigate the available trails, by using SNA Database that contains the weight of relations to determine if two trails are related considering attributes, such as amount of funds involved, geography, affinity of participants, time, method for laundering, relation.

The module “SNA” analyzes the end and start points (“EndBoss” and “StartBoss”) of discovered transaction sequence (trail) to discover any broken parts of such none-discovered trail. This analysis includes assessing the social relationships between the endpoints (“EndBoss” and “StartBoss”) of trails with each other, using the DB created that includes different level of relationships. The participants and bosses of money laundering trails may change, but the key players stay the same and they swap roles (Mehmet et al., 2013).

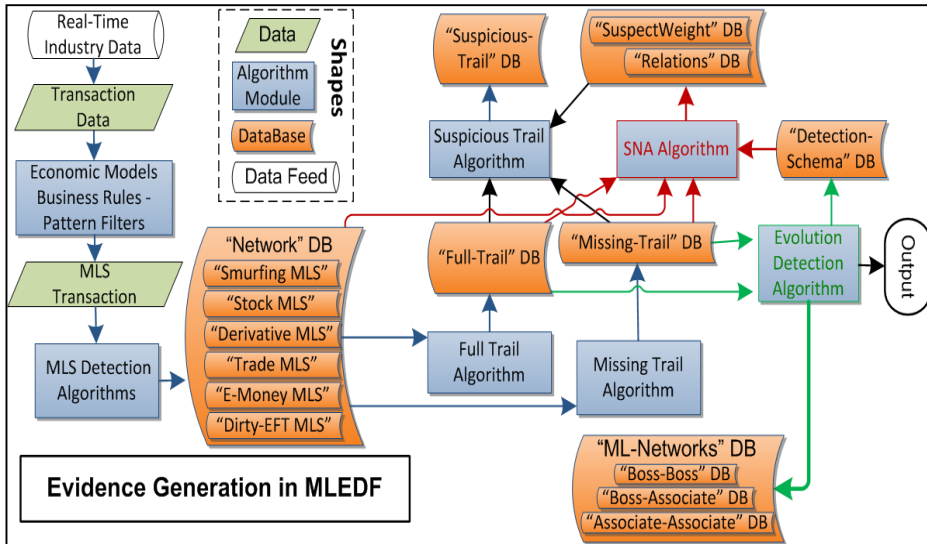


Figure 3 The Process of Generating Evidence Data in the MLEDF

### 3.3 Input from Algorithmic Modules and their Databases into the “SNA” Module

This section describes modules called inside the MLEDF and feed their output (DB) as an input to the “SNA” module.

1. *“Full-Trail” module*: We create the long trail “Full-Trail” of complex MLS series that span over many countries. A “Full-Trail” is a concatenated sequence of related ML schemes acts as a whole to transfer money from one MLS to another. A “StartBoss” of a “Full-Trail” is the “StartBoss” of the first MLS in the series of MLS that constitute the “Full-Trail”. Whereas the “EndBoss” of a “Full-Trail” is the “EndBoss” of the last MLS in the MLS series that constitute the “Full-Trail”. We detect the “StartBoss” and the “EndBoss” of the Full-Trail, along with intermediary bosses of linked schemes. The “StartBoss” is the earliest known launderer (that we have proven evidence for) that initiates the sequence of ML transactions. The “EndBoss” is the final launderer that withdraws the funds or transfers them using remittance (Hawala) systems that do not keep any financial records. We start with the “EndBoss” and compute the laundering path towards its beginning. Then we follow all possible paths that originate at the detected launderer “StartBoss” and link to another launderer. During concatenation of the schemes we consider the amount of funds involved, geography, affinity of participants, time, relation, and method for laundering.

Table 1 Sample Output of the Full-Trail (Up to maximum of 30 related MLS networks)

Networks	TrailID	Duration	Withdraw	Amount	Start Boss	End Boss
24, 51, 67, 92, ....	1932	56 Days	Yes	988,000	Boss 756	Boss 17
2, 15, 98, 126, .....	72468	99 Days	No	1,213,234	Boss 29	Boss 592
415, 783, 999, .....	97246	92 Days	Yes	1,050,230	Boss 324	Boss 75

2. *“Missing-Trail” module:* We create short trails that do not exceed the depth of three consecutive MLSs, or three levels depth of MLSs. We assume that Missing-Trail is a premature Full-Trail with broken parts and missing evidence. Therefore we capture such shorter trails for “Suspicious-Trail” analysis by saving them in the “Missing-Trail” DB.

Table 2 Sample Output of the Missing-Trail (Do not exceed 3 related MLS networks)

Networks	TrailID	Duration	Withdraw	Amount	Start Boss	End Boss
14, 219,921	1232	16 Days	No	23,234	Boss 56	Boss 151
2452, 315	1208	29 Days	No	90,165	Boss 170	Boss 882
405, 7831	97246	19 Days	No	200,230	Boss 884	Boss 975

3. *Evolution-Detection module:* We analyze the input feed from “Full-Trail” and “Missing-Trail” algorithms, and generate DB “ML-Networks” and DB “Detection-Schema”. The DB “ML-Network” contains three DBs of “Boss-Boss”, “Boss-Associate” and “Associate-Associate”. The three DBs reflect the all discovered pair relationships among bosses and associates of MLSs and trails. An associate is a participant of the MLS who facilitates the success of MLS, such as the deposit makers in smurfing or the stock broker in stock based MLS. The DB “Detection-Schema” contains statistical information about the evolution of the ML trail, such as information of average cost and average number of associates used in each MLs of the sequence of MLSs in the Full-Trail. Table 3 shows the shortened output of the detected schema of the real-life case.

Table 3 Sample Shortened Output of Detection-Schema

Detection Time	Type	SubType	Network ID	Location	StartBoss	EndBoss
20120915114	HiLo	Hi	2213	USA	Boss756	Boss 17
20120819139	Stock	LowSale	9786	Germany	Boss 324	Boss 75
Associates	Cost	Amount	Withdrawal Time	Start Time	End Time	
A, B, G, U	25,000	1,825,000	20120915114	20120830	20120915	
N, O, W, Y	14,700	1970,000	20120819124	20120725	20120819	

The “SNA” module will use the content of the DBs produced by the three evidence generation algorithms as an input. The contents of the output generated from the three modules listed in above tables will be saved into three DB named “Detection-Schema” DB, “Full-Trail” DB, and “Missing-Trail” DB. Every algorithm will create a DB with the same name of the algorithm name. Additionally, the DB “Network” associated with each MLS type will also feed into the SNA module.

### **3.4 The Components and Output of the “SNA” Module**

The SNA module will generate the two Databases as outputs. The “SuspectWeight” Database contains the weight of all different relations detected in the MLEDF. The “Relations” Database contains the record of business and family relations among pairs based on the assumption that we have access to such records. The method to calculate the calculation is explained in the next section. The “SNA” module continuously updates the “SuspectWeight” with the value (score) of existing relations among the entities existing in all the DBs created in the MLEDF.

Table 4 Components of the Two DB Output of SNA

**RelationshipDB** ( hash string, #"time" timestamp, type string, person1 string, person2 string) PRIMARY KEY (hash, #"time", type) USING BTREE;

**SuspectWeightDB** (hash string, UniqueTrailBosses long, UniqueTrailAssociates long, UniqueMissingTrailAssociates long, SchemaBosses long, SchemaAssociateBoss long, SchemaAssociate long, family long, business long, weight long) PRIMARY KEY (hash) USING BTREE;

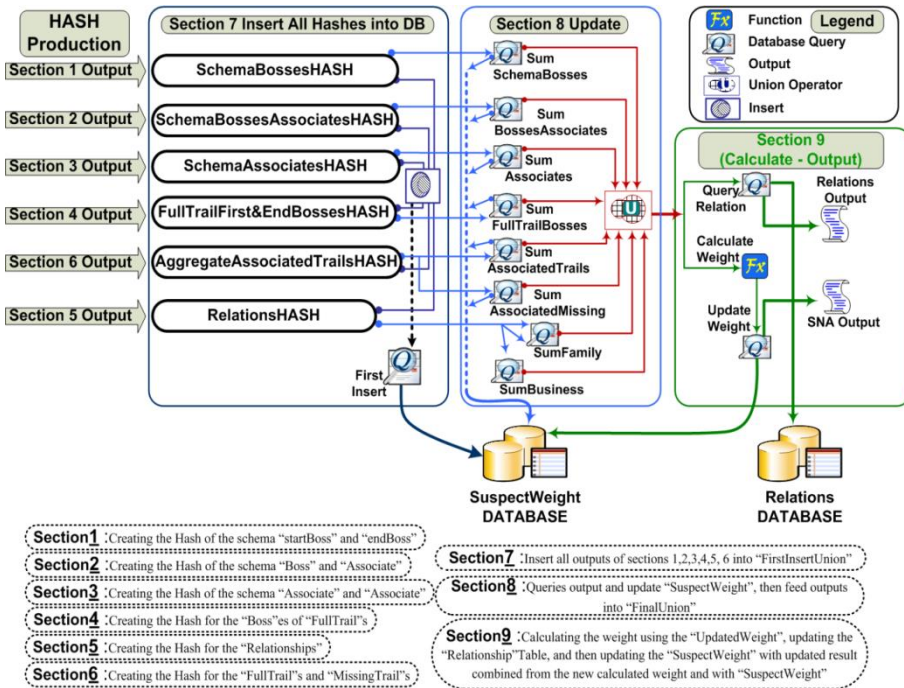


Figure 4 Sections of the Social Network Analysis Module

The SNA module contains nine sections to continuously update the two Databases. Section 1 through 6 creates hashes of various relations. The sections 7 through 9 update, query, and calculate SNA weight of family, business, and various ML relations derived in earlier stages. The hash of binary relations, that involves pair of entities, is used as the basis to calculate the accumulative relations of every type. The sections list is as follows:

- a. Section 1: Creates Hashes for all "StartBoss" and "EndBoss" relations, we refer to it as "Hash SchemaBosses". The "StartBoss" and "EndBoss" are unique to a MLS.

- b. Section 2: Creates Hashes for all detected “Boss-Associate” relations we call “HashBossesAssociate”.
- c. Section 3: Creates Hashes for all detected “Associate-Associate” relationships. This hash represents the combinations of relationships among the associates of the same MLS, even if they do not interact/transact with each other directly.
- d. Section 4: Creates Hashes for “StartBoss”-“EndBoss” pairs of Full-Trails, we call “HashFullTrailBosses”.
- e. Section 5: Creates Hashes for “Relationships” among socially or business-wise related pairs.
- f. Section 6: Creates Hashes for the associated (related) Full-Trail’s and Missing-Trail’s, we call “AggregateOfAssociatedTrails”. This hash contains a pair of TrailIDs, which are unique IDs assigned to each detected trail. The hash is used to relate trails by their TrailIDs.
- g. Section 7: Inserts all the outputs (hashes) of Sections 1, 2, 3, 4, 5, and 6 into “SuspectWeight” DB, using “FirstInsert”.
- h. Section 8: Queries the “SuspectWeight” for the continuously updated outputs (hashes of Sections 1-6), then, feed the updated hashes values into last section.
- i. Section 9: Calculate the accumulative weight of relations using the “UpdatedWeight”, updating the “Relationship” table, and then updating the “SuspectWeight” with updated result combined from the new calculated weight and with “SuspectWeight”. The formula (method) used in this stage to sum up the accumulative weight, by adding the hash relations from Sections 1 through 6, is explained below in the next section, This final and updated weight will be used in the “Suspicious-Trail” module to link trails among each other.

### **3.5 Social Network Analysis Algorithm**

The “Social Network Analysis” algorithm computes the weight for different relationships involving the bosses and associates of MLS and trails. The values of the weights are chosen based on the importance of relation in a scheme, that is to say a relation of certain type is not treated equally as a relation with less importance. Also the margin of weights chosen allowing an iteration of certain relation to be equal in weight value to another relation with a higher weight, for example two “Boss-Associate” relations is value wise equal to one “Boss-Boss” relation.

A relationship weight is defined for each possible associate couple. The larger the weight, the more likely the relationship between two entities to occur. Weight is calculated by adding parameters for each of the corresponding events; therefore, the result is considered as the relationship weight.

- a. For each detected schema, 10 will be added to start/end boss couple, 5 for each boss/associate combination, and 1 for each associate/associate non-repeating combination.
- b. For each missing trail, 15 will be added to each associate non-repeating combination.
- c. The full trails will add 20 to each associate combination and 25 to the start and end boss.
- d. Other strong relationships are also counted, family ties will add 250 to the couple, and each business relationship will add 250 to the couple.

---

```

1 SET BUSINESS as 250; FAMILY as 250; FULL_BOSS as 25;
  FULL_ASSOCIATE as 20; MISSING_ASSOCIATE as 15;
  SCHEMA_BOSS as 10;
2 SCHEMA ASSOACIATEBOSS as 5; SCHEMA_ASSOCIATE as 1;
  FUNCTION String HASH (String person1, String person2)
3     {return concatenate(sort(person1,person2))};
  STREAM DetectionInputStream DetectionSchema detectedMLS;
  STREAM RelationshipInput RelationshipSchema relationship;
  STREAM MissingTrailInputStream MissingTrailSchema missingTrail;
4 STREAM FullTrailInputStream FullTrailSchema fullTrail;
5 STORE hashRelations IN hashAndRelationsNumberMemoryDB;
  UPDATE hashAndRelationsNumberMemoryDB as H
    SET suspectSchemaBoss++
6     WHERE H.hash == HASH(detectedMLS.startBoss,
    detectedMLS.endBoss);
  UPDATE hashAndRelationsNumberMemoryDB as H
    SET suspectSchemaAssociateBoss++
7     WHERE H.hash == HASH(detectedMLS.associate,
    detectedMLS.endBoss);
  UPDATE hashAndRelationsNumberMemoryDB as H
8     SET suspectSchemaAssociateBoss++
    WHERE H.hash == HASH(detectedMLS.associate,
    detectedMLS.startBoss);

```

---

---

```
FOR EACH detectedMLS.associates as assoc1
  FOR EACH detectedMLS.associates as assoc2
9  UPDATE hashAndRelationsNumberMemoryDB as H
    SET suspectSchemaAssociate++
    WHERE H.hash == HASH(assoc1, assoc2);
FOR EACH fullTrail.associates as assoc1
  FOR EACH fullTrail.associates as assoc2
10 UPDATE hashAndRelationsNumberMemoryDB as H
    SET suspectFullAssociate = suspectFullAssociate++
    WHERE H.hash == HASH(assoc1, assoc2);
11 UPDATE hashAndRelationsNumberMemoryDB as H
    SET suspectFullBoss = suspectFullBoss++
    WHERE H.hash == HASH(fullTrail.startBoss, fullTrail.endBoss);
FOR EACH missingTrail.associates as assoc1
  FOR EACH missingTrail.associates as assoc2
12 UPDATE hashAndRelationsNumberMemoryDB as H
    SET suspectSchemaBoss = suspectMissingAssociate++
    WHERE H.hash == HASH(assoc1, assoc2);
UPDATE hashAndRelationsNumberemoryDB as H
13 SET suspectBusiness = suspectBusiness + 1
    WHERE H.hash == HASH(relationship.person1, relationship.person2)
    AND relationship.type == "BUSINESS";
UPDATE hashAndRelationsNumberMemoryDB as H
14 SET suspectFamily = suspectFamily + 1
    WHERE H.hash == HASH(relationship.person1, relationship.person2)
    AND relationship.type == "FAMILY";
SELECT H.hash,
  (FULL_ASSOCIATE*H.suspectFullAssociate +
  FULL_BOSS*H.suspectFullBoss +
```

---



---

<pre>MISSING_ASSOCIATE*H.suspectMissingAssociate + SCHEMA_ASSOCIATE*H.suspectSchemaAssociate + SCHEMA_ASSOCIATEBOSS*H.suspectSchemaAssociateBoss + SCHEMA_BOSS*H.suspectSchemaBoss + BUSINESS*H.suspectBusiness +FAMILY*H.suspectFamily) as <b>WeightOutputStream</b> FROM hashAndRelationsNumberMemoryDB as H;</pre>
---

---

Query 1 The Social Network Analysis Algorithm

In steps 1 and 2, we define the constants associated with the different weights and the hash functions. In steps 3 and 4, we create the input feeds and local (temporary) MemoryDB. In steps 5 through 8, we create the hashes of “Boss-Boss”, “Boss-Associate”, and “Associate-Associate” of MLs. In steps 9 through 11, we create the same hashes of Full-Trails. In steps 12 and 13, we create the hash for family and business “Relations”. In step 14, we calculate the WeightOutput of a hash H.

### 3.6 The “Suspicious-Trail” Analysis Module

Using the “SNA” module, the “Suspicious-Trail” module produces a new trail that contains the full path of an evolution, after making a scientific assumption of a possible relation between two or more “Full-Trail” and “Missing-Trail” lists. This module continuously calls the “SNA” module to fetch the social network connections of individuals, peer-groups, and money laundering trails. The sub-module “SNA” is used to derive the associated suspicious trails based on SNA techniques to calculate the link weight (ML relations found in all MLEDF DBs) and attribute (business and family relations) weight among trail actors. The table “Weight” that is used within the “Suspicious-Trail” module is completely constructed with data output from running the “SNA” module. The generated evolution patterns and strategies are collected into the “Suspicious-Trail” Database.

### 3.7 Suspicious-Trail Algorithm

The “Suspicious-Trail” algorithm searches for missing and hidden links among ML trails. The analysis starts with any new “Full-Trail” with no more than 30 networks (step 5). Then, the current “Full-Trail” is matched with each trail that complies with (step 6):

- a. Having  $\pm 10\%$  of the current “Full-Trail” amount.
- b. A time window of 90 days between both trails’ timestamps.
- c. Location of first network of current “Full-Trail” is the same as the location of the last network of possible “Full-Trail” match.

- d. Weight between current “Full-Trail” first boss and possible match last boss is larger than 1000.
- e. All matches are then treated separately as study cases and related to the current “Full-Trail” forming a “Suspicious-Trail” (steps 7 and 8).

<b>1</b>	SET NetworkLimit as 30; Similar_FundPercentage as 0.1; SET DayWindow as 90; SET WeightRelation as 1000
<b>2</b>	STREAM <b>FullTrailInputStream</b> FullTrailSchema Trails;
<b>3</b>	STREAM <b>SNAOutputStream</b> WeightSchema weight;
<b>4</b>	STORE Trails IN <b>TrailsMemoryDB</b>
<b>5</b>	SELECT Trails FROM Trails WHERE (Trails.lenght <= NetworkLimit);
<b>6</b>	SELECT Trails, db as matchTrails FROM Trails as m, TrailsMemoryDB as db WHERE (m.finalAmount * Similar_FundPercentage < current.finalAmount) AND (m.finalAmount > db.finalAmount* Similar_FundPercentage) AND
<b>7</b>	((m.detectionTime-db.detectionTime) <= days(DayWindow)) AND (lastelement(db.networks).location == firstelement(m.networks).location));
<b>8</b>	SELECT CONCATENATE(t,m) as <b>SuspectedTrailOutputStream</b> FROM Trails as m, matchTrails as t, weight as w WHERE w.hash == sort(t.startBoss,m.endBoss) AND w.weight >= WeightRelation; SELECT CONCATENATE(m,t) as <b>SuspectedTrailOutputStream</b> FROM Trails as m, matchTrails as t, weight as w WHERE w.hash == sort(m.startBoss,t.endBoss) AND w.weight >= WeightRelation;

Query 2 The Suspicious-Trail Analysis Algorithm

In step 1, we define the constants associated with limits set to relate and compare trails. In steps 2 through 4, we create the input feeds including the SNA-generated Weight DB, and local (temporary) DB TrailsMemoryDB. In

step 5, we filter the analyzed trails that do not exceed 30 levels. In step 6, we compare and link trails that match based on criteria listed above. In steps 7 and 8, we concatenate the matching trail with another trail, either trail or matching trail leading the generated outcome.

## **4. EXPERIMENTAL EVALUATION**

It hard to obtain real-life data in the domain of ML, where one can find some samples used to explain complex real-life cases. Therefore we approached several organizations to collect sanitized real-life cases that validate the testing of MLEDF, meeting the requirements imposed by the organization that provided the sanitized cases. Our case studies are based on data provided from the organization we refer as Trusted Third Party (TTP), which is legally allowed to collect information and track records of financial exchange. The identity of the TTP cannot be disclosed due to a Non-Disclosure Agreement. The sanitized cases were provided on the basis of having the MLEDF also tested in the infrastructure of TTP.

### **4.1 Experimental Setup**

Using the real-life dataset, we generate a larger dataset that contains different levels of random transaction using a module we implemented using Java. We used a template from real-life cases to generate the synthetic data that is similar to those cases by selecting a subset of  $t$  real-life cases to create more samples and develop new patterns, based on criteria such as preserving duration flexibility, geography variation, multitude of fund transferred, crowded trails, trails with low funds, complex instruments such as derivative products, continuous transition from one financial sector to the other, splitting a transaction with large fund into many connected small funded MLSs, etc. Once we generate artificial data sets we unite all the databases to create a large dataset to be inserted into the MLEDF for validation and testing. All the “endBoss” and “startBoss” of trails and generated MLS series are compared against the detected ones of MLEDF. The same test was repeated with inserted random patterns of small trails and MLSs in some interval to confuse the MLEDF and test the false positive rate (FPR) and false negative rate (FNR). By adding a combination of randomly generated MLS series we imitate the daily production environment of stock brokerage and a retail bank. The real validation test for MLEDF is accomplished by assessing its ability to detecting patterns with accuracy in a noisy environment, similar to the real life transactions that are filled with noise.

### 4.2 Using Real-Life Cases in the Validation Test of the MLEDF

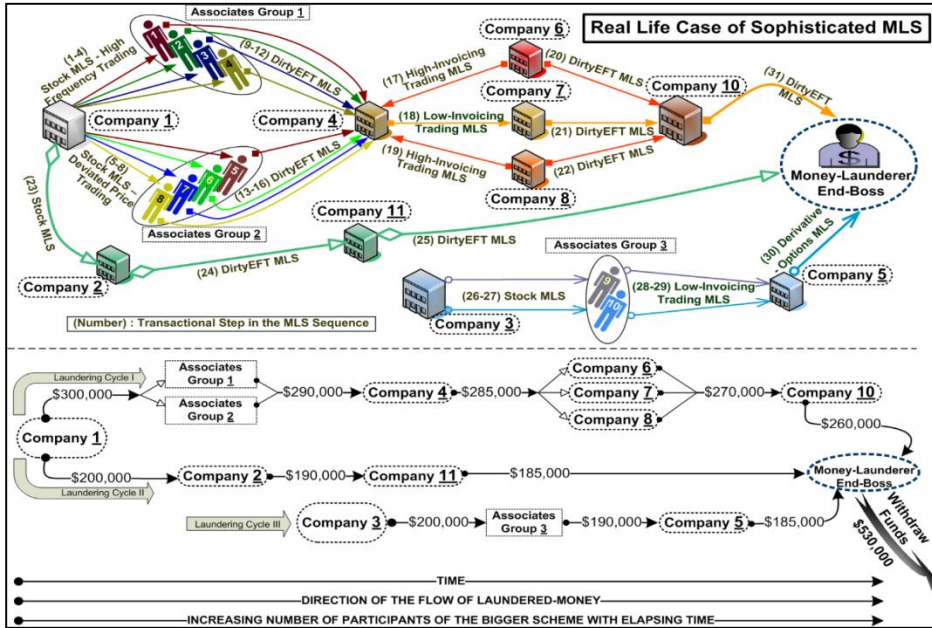


Figure 5 Real-Life Case of Evolved and Sophisticated MLS

Our test case spans over 5 countries involving 11 companies, 10 associates, and 8 innocent entities. As shown in Figure 5 and Table 5, the scheme has 3 different full trail cycles. The laundered amount is lower than the amount transferred by the “StartBoss” of the cycle. The amounts in the case only reflect the amount laundered, either by gaining and losing, or by means of transferring the value. The amount does not represent the full amount of the transaction, which is higher than the amount laundered. The masterminds of the scheme are Company1 and “EndBoss”, based on the information provided to us by TTP. The sub-cycles of the case occur independently of each other and each has different rounds. Each round in each sub-cycle is in tandem with other rounds of the same sub-cycle. The information of individuals and companies, locations, and dates are all sanitized.

Table 5 The Subcycles and Rounds of the Full-Trail of the Real-life Case

Cycle StartBoss	Cycle EndBoss	Participants	Rounds	Laundered Amount
Company1	Money Launderer	15	5	<b>\$260,000</b>
Company1	Money Launderer	4	3	<b>\$185,000</b>
Company3	Money Launderer	5	3	<b>\$185,000</b>

### 4.3 Experimental Evaluation

We introduced a three phase testing prototype to examine MLEDF and detection algorithms. All of the three phases focused on testing and validating the components of MLS, Full-Trails, and Suspicious-Trails. The first phase focus on testing all components and the other tests focus on Full-Trail and Suspicious-Trail components.

Table 6 Used Testing and Validation Methodology

Test-Validation Type	Patterns Used	Pattern Generation Method
Test (I)Without Noise	Single MLS,	StreamBase-Generated single MLS
	Missing-Trail,	StreamBase-Feed Pair of MLS
	Full-Trail	Feed Full-Trail's from Real-life Cases
Test (II)Subtle Noise	Entities,	Inject same entities into Full-Trail
	Transactions,	Inject subtle transactions into Full-Trail
	Single MLS	Inject similar MLS into same Full-Trail
Test (III)Controlled Data	30LDeep Full-Trail	Create 30 levels vertically deep trails
	20LDeep Full-Trail	Create 20 levels vertically deep trails
	10LDeep Full-Trail	Create 10 levels vertically deep trails

Table 7 Defining False Negative Rate for Each Test Phase

Test	Patterns Used	FNR
Test I	Single MLS,	Detected MLS list is less than the input list
	Pair MLS,	Detected MLS pairs is less than the input list, or MLS pair is detected as single MLS
	Full-Trail	Detected Full-Trail list is less than the input list, or Full-Trail is detected as Missing-Trail's (shorter trails) and single MLSs
Test II	Entities,	Missed detection of MLS, because similar participants injected
	Transactions,	Missed detection of MLS, because identical transactions injected
	Single MLS	Missed detection of trail, because similar MLS injected
Test III	30L-Deep	30L-Deep is missed in detection, and it will cause to be detected as (FPR) other level deep combination of Full-Trail (less than 30L), Missing Trails and MLS
	20L-Deep	Missed detection, and causing generation of FPR of Full-Trail (less than 20L), Missing Trails and MLS
	10L-Deep	Missed detection, and causing generation of FPR of Full-Trail (less than 10L), Missing Trails and MLS

Table 8 Defining False Positive Rate for Each Test Phase

Test	Patterns Used	FPR
Test I	Single MLS,	Not possible as MLS is either detected or missed, as there is no noise
	Pair MLS,	Not possible as MLS pair is either detected or missed, in no-noise data
	Full-Trail	Full-Trail is not captured, instead MLS pairs, MLS triple, or shorter Full-Trails are captured in lieu of the Full-Trail
Test II	Entities,	Detect MLS with different participants (Associate and/or Boss)
	Transactions,	Detect MLS with different participants (Associate and/or Boss)
	Single MLS	Missed detection of the actual Full-Trail. Instead of that MLS, Missing-Trails and shorter Full-Trails that form the actual Full-Trail captured
Test III	30LDeep	Full-Trail of desired depth (10L, 20L, 30 L) is not captured, instead a combination of Full-Trail of less depth, Missing-Trail, and MLS are captured in lieu of the actual Full-Trail
	20LDeep	
	10LDeep	

**Test without noise:** This is designed to test every module of MLEDF, including detection algorithms and trail analysis modules. These tests evaluate the FPR and FNR by comparing the results of the test with the data feed that contains the patterns of single MLS, pair of MLSs, and Full-Trails. The desired result is to have a list of the validation result identical to the list in the data feed. We test the efficiency to keep up with the speed of the data feed by using the time window feature in the StreamBase (StreamBase 2012, 2014). By setting the time window to glide over only one event at a time tick in the StreamBase system, we required the detection algorithms to be fast at normal speed of one event at one time tick of the CEP system. By design, algorithm that cannot attain the speed of event production will not be able to capture MLS events or the Full-Trail, thereby generating false negatives.

Each of the six detection algorithms are tested with its own dataset feeds in order to verify that we detect without FPR and FPR. The algorithm-specific dataset feed is generated using the built in feed generator working with our pattern specific event generator. Afterwards, we tested the “Missing-Trail” by feeding linked pairs of MLSs into the MLEDF. The linked/related pairs are randomly selected from the set of six types of MLS. As mentioned, any pair of linked MLS will make it to “Missing-Trail” and not into “Full-Trail”, due to the required depth. Moreover, we finally tested the detection and evolution of “Full-Trail”s by feeding trails generated using the various laundering strategies of the real-life cases.

The process of creating the “Full-Trail” will start with creating an MLS type out of the six MLS types of Smurfing, Trading, DirtyEFT, Stock, Derivative, E-Money. Once the selection of first MLS is made, we create the linked MLS series based on considerations such as geography, money-amount, time, complexity, difficulty of tracking. The trails were created considering different levels of criteria, the randomization of the criteria is uniformly distributed. The Full-Trail feeds were created by the generator that does not exceed 10 levels of depth of linked MLSs. The trails are either a variant or a subsection of one of the real-life cases that are similar in terms of complexity and people involved to the case explained previously.

At the normal speed, of one event at one time tick of the CEP system, the test result is zero for the false positive rates and false negative rates. It is highly improbable to get a false positive trail due to the business rules that define them, and due to the accuracy and granular level of linking transactions. We did not get any false positive schema in the MLS tests, due to the synthetic nature of the data. When we increased the speed of the feed of data generated to 10 times and 100 times the normal speed, we observed a FPR and FNR in the objects detected (Full-Trail). Increasing the speed of data-feed processing did not produce FPR and NFR for single MLS, but it produced FPR and FNR for pair MLS at speed 100X. The term “object” in the graph refers to the three

different patterns of single MLS, pair MLS, and Full-Trail in the proprietary test of the specific object (Object in the first pattern test refers to the first pattern single MLS, in the second to MLS pair, in the third to Full-Trail). The values of FRP and FNR reflect the number of falsely detected objects.

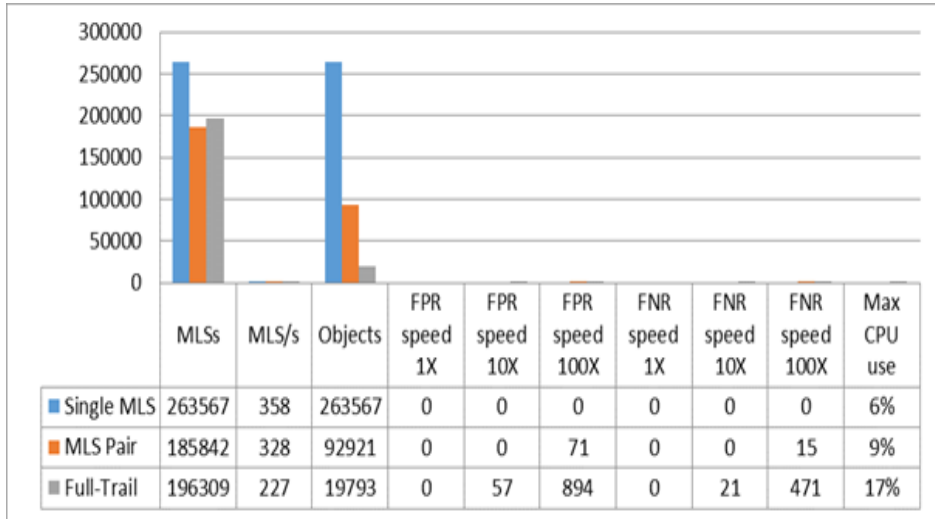


Figure 6 Details and False Detection Results of Test I

**Test with subtle noise:** This is the most relevant test of the accuracy of our detection algorithms. The goal is to mislead the detection algorithms to generate both a false positive and false negative, with the use of subtle synthetic data. The test has three separate phases: injecting the scheme participants, injecting subtle transactions, and inserting similar MLS. A subtle transaction means an identical transaction with  $\pm 5\%$  of an actual transaction amount in a MLS. A similar MLS means an identical MLS with the same set of participants but with the MLS value is  $\pm 10\%$  of the laundered amount of the MLS. The injection speed was performed at normal processing speed, 10 times faster speed, and 100 times faster speed. The test of injecting transactions and MLS is setup considering each MLS type. For example, in the test of smurfing we create only smurfing MLS and smurfing transactions that can extend vertically up to 20 levels of depth and horizontally to 30 levels of depth. When generating the MLSs, our measures vary based on the MLS. We do not use artificially created none-real life cases. For example, we did not use a smurfing MLS with 100 levels deep, as that is uncommon and impractical to launder money using such MLS. We do not inject other MLSs into the injection testing of specific MLS. However, in the Full-Trail testing we inject all the types of MLS. By design of full-Trail it is required that we related different types of MLS under the same Full-Trail.



As it can be seen from Figures 7-9, the test produced low FNR and low FPR for transaction and MLS injection when the phases were executed at normal processing speed. Those rates increased in the phases when tests were executed at faster processing speed. One way to imitate the data feed pace of real production environment is to run the CEP tests at faster pace, which means overloading the system with processing and analytics while attempting to keep pace with the data feed. The goal was to evaluate the effectiveness of “Full-Trail” detection when the system absorbs data at a higher rate while performing the analysis. Due to the design methodology of detection algorithms and the complexity of the business rules of MLS detection, their false detection rates stayed at low levels even with injection similar transactions and MLSs, at a higher data-feed speed.

Meeting the design principles, the “Full-Trail” and “Suspicious-Trail” results remained at low rates for both false positive and false negative. Therefore, all the subtle single MLS created with our injected data ended in the “Missing-Trail”s, where they do not exceed the depth of 3 consecutive MLSs. Among the reasons for such success in trail analysis and avoiding any negative impact are the following: (1) Designing MLEDF in such a strict and granular method, especially for matching the MLSs within the trails; (2) Using the SNA in the trail analysis algorithms; and (3) Adopting the criterion of following the direction of the money flow. MLS is not expected to terminate with funds remaining in the account. The money must flow in some direction in order to be laundered, or must be withdrawn by the launderer.

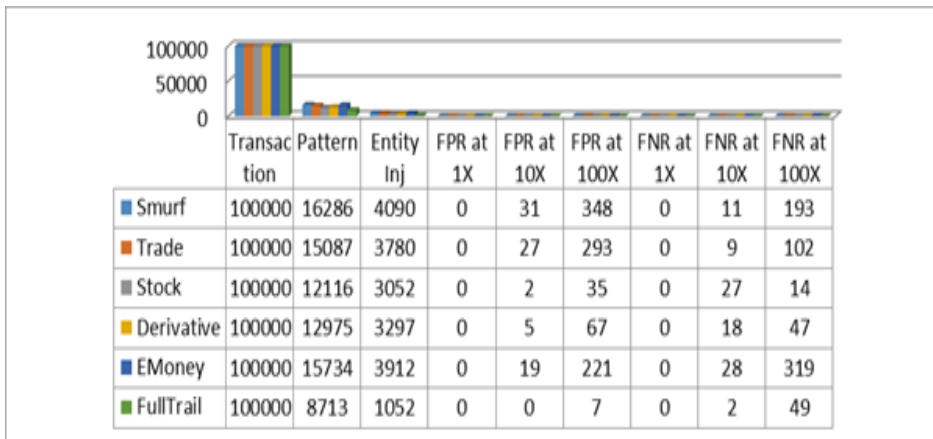


Figure 7 Details and False Detection Results of Test II-Entity Injection

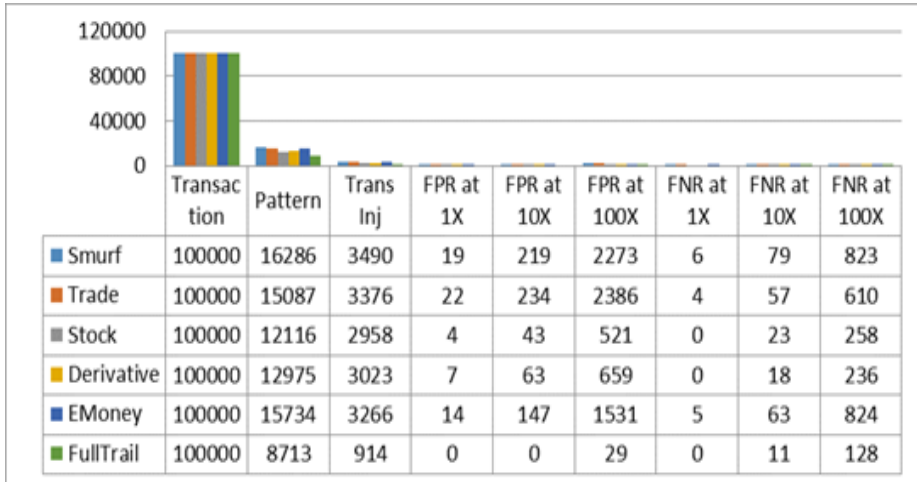


Figure 8 Details and False Detection Results of Test II-Transaction Injection

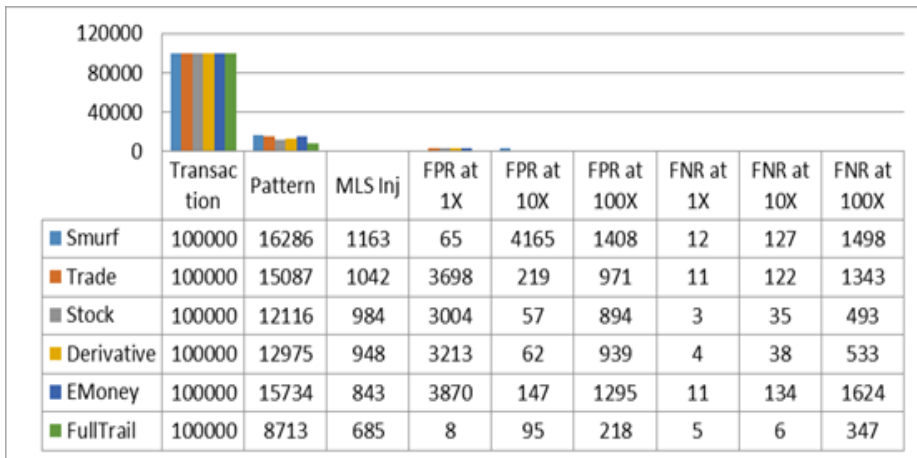


Figure 9 Details and False Detection Results of Test II- Similar MLS Injection

**Test with longer synthetic full-trails:** This is the hardest level of performance-testing of the system and accuracy-testing of the detection algorithms. The dataset is permuted over a repository of the different real-life cases. Afterwards, the dataset is combined with randomized MLS to generate deep vertical levels of “Full-Trail” and “Suspicious-Trail”. The randomization follows the same principles we used in Test II, the injection testing. The test is designed to assess the performance of MLEDF in capturing real-life data and analyzing them on the fly. The desired test result is to generate low FPR and FNR. The test module generates all synthetic data from real-life cases. The test is as similar as it can be to real-life scenarios, considering that there are limited ways to manipulate a MLS. The test program functions as follows: (1) Set a

trail depth. The program enters a loop and builds a trail by choosing a first scheme from of each MLS type at random, as it was described in Test I in building the Full-Trails; (2) The loop continues by creating an MLS that can be linked by funds, time, location and complexity wise to the current MLS. We repeat the step above with the exception of not creating any Smurfing MLS for the rest of the levels; (3) The permutation continues until we reach the last level, where we always choose an MLS of type DirtyEFT with withdrawal, in order to generate the trail termination point, as by definition a trail will end with withdrawal; (4) The test repeats the process of trail generation forever, and at the maximum possible speed; and (5) The testing module saves the arrival time of the last DirtyEFT and subtracts that from the build time of the trail. Thereby, we obtain the difference in Milliseconds, which represents the time duration for trail processing.

The data was generated for worst-case scenarios. By doing so we ensure that the generated data is more complex and that the performance is evaluated only in most resource consuming cases. Displayed results represent the performance of data generated without any repetitive bosses or associates. Hence, the dataset is consumes a significant number of resource

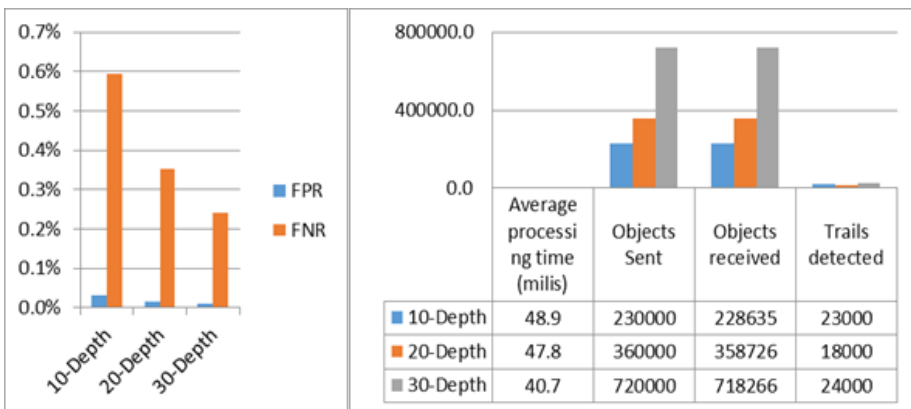


Figure 10 False Detection Results and Details of Test III

### 3.4 Data Characteristics

We introduced six MLS types with different subtypes for each sector. The combination of schemes is novel in its entirety as they were driven from real-life cases. The novelty lies in: (1) creating patterns from real life cases using CEP system; (2) developing a software that can read patterns from real-life or synthetic cases and evolve it based on criteria we implemented; and (3) attempting to link the networks of from those cases to produce different MLS variations that involve all well-known MLS.

In the first test, we used the StreamBase (SB) simulator (StreamBase 2012, 2013) and MLS DB to feed data into the application. In the tests of the MLS

pair and “Full-Trail”, the simulator retrieves the DB data and generates samples. In the single MLS test, we feed the DB data into the simulator to generate samples from a pool of a specific set of MLS. The “Full-Trail” dataset contains variants and subsections from five real-life cases. The dataset was in CVS format, and contained 292,000 MLS records that constituted a total of 12,000 “Full-Trail” records. The dataset contained 1.4 million transactions. In the second test, we used a modified version of the dataset that contains subtle transactions and identical MLSs. We injected 20,000 identical MLS and 95,000 subtle transactions using the simulator. In the third test, we implemented a Java program to stream the generated data into the MLEDF.

### 4.5 Testing Performance

In order to test performance using the environment described above, we executed three major tests over three different data sets. The third simulation test was the most crucial test, as the data feed resembles the data feed seen on a daily basis in a bank or in a brokerage house.

The processing ratio is ideal, obviously workable to be faster. The trail processing time decreases with the deeper trails because there are less MLS per object in this case. However, it is a good indicator because it means that in more healthy situations, the system will be faster in responding to troubles. The processing time both for objects and MLSs is set to be ideal on 20-depth trails. This implies that the system is more suitable to trails of that particular depth. With our assessed speed to get ~0.5 objects/millisecond processing, and for such a system, this is significant. While the processing ratio can be very subjective to the number of markets, for banks and other feeds connected to the system, a productive version of a system like this will need to be scaled in order to meet that ratio.

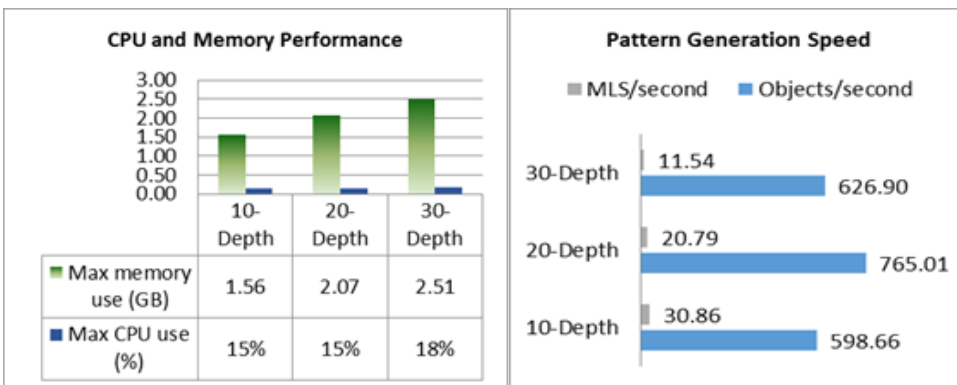


Figure 11 Performance Test Results

#### **4.6 The Validation Statement**

We used the SB Studio Feed Simulation platform for performing our tests during the validation: (1) to connect it to module generator of Test III to retrieve the deep Full-Trails; (2) to connect it to the data-feed we used in Test II; and (3) to create noise-free pure data in Test I. Although there are higher performance opportunities promised by vendors, SB in particular, we observed that our queries reveal reasonable outcomes in terms of performance tests, even using this none-enterprise test environment. We also observed that the queries result in reasonable accuracy values given the specially crafted synthetic data sets. Given the data set, we successfully determined a reasonable window size for window-based queries. Using attributes, we could also successfully converge the evidence outcomes by tuning the force of time, property, and key-based patterns directing those queries.

#### **RELATED WORK**

The system created by FINCEN AI System (FAIS) (Senator et al., 1995, 1998) is a similar system in terms of the concept of detecting MLS. The drawback of the system is that it does not capture live data (i.e., running data connected to banking systems) and requires the involvement of an expert in the link analysis, and tying MLS and transactions. Whereas our system does not need the involvement of an expert and captures live, as well as feeds the data into the SB engine. The KDPrevent system (Jacobs et al., 2003; Kuns et al., 2004) is the most private product that is similar to our product in terms of the logic of including the background of the transactions of individual and groups. The KDPrevent system is also based on data mining techniques that is not real-time and necessitates the involvement of experts. Gunestas, et al. (2010) conducted a similar study in the financial transaction forensics, with a focus only on Ponzi schemes. The framework only captures one form of transaction only from the web services transactions. Our framework can be accepted as a continuance to (Gunestas et al., 2010) for MLS and accepting all forms of transactions, including the banking transactions.

#### **6. CONCLUSION**

We have created a framework to detect the evolution of MLS and implemented a system to include SNA for detecting and linking related ML networks. The linkage will function properly even when all evidence is unavailable. We defined the choreographies that could be used to detect the evolution of the sophisticated MLS. We have shown how to detect and capture the evolving and complex trails of MLS using SB. Although our choreographies only specify well-known money laundering schemes, the framework can be updated with business rules to capture any form of other MLS that can be mined from repositories of financial transactions. Our ongoing work addresses the extension of our method in developing an online warning system that detects

MLS that appear legitimate from an abstract view, but are illegitimate from the detailed view. Currently, we are working to produce algorithms to prevent transactions from a sequence of financial transactions, based on our detection system and proprietary scoring system.

### REFERENCES

Gunestas, M., Wijesekera, D., & Singhal, A. (2008). Forensic web services. *Fourth Annual IFIP WG 11.9 Conference on Digital Forensics*.

Gunestas, M., Mehmet, M., & Wijesekera, D. (2010). Detecting illegal business schemes in choreographed web services: The Ponzi/Pyramidal case. *Sixth Annual IFIP WG 11.9 Conference on Digital Forensics*.

Jacobs, L., & Wyss, R. (2003). KDPprevent: White paper: intelligent detection of money laundering and other financial crimes. *KDLabs Reports 2003*.

Kunz, B., & Peter, S. (2004). KDPprevent: detecting money laundering activities. *KDLabs Reports 2004*.

Liu, X., Zhang, P., & Zeng, D. (2008). Sequence matching for suspicious activity detection in anti-money laundering. *Proceedings of the IEEE ISI 2008 PAISI, PACCF, and SOCO*.

Mehmet, M., & Wijesekera, D. (2010). Ontological constructs to create money laundering schemes. *Semantic Technologies for Intelligence, Defense, and Security Year 2010*.

Mehmet, M., & Wijesekera, D. (2013). Detecting the evolution of money laundering schemes. *Ninth Annual IFIP WG 11.9 Digital Forensics*.

Schwartz, D., & Rouselle, T. (2008). Using social network analysis to target criminal networks. *Trends in Organized Crime Year 2008*.

Senator, T., Goldberg, H., Wooton, J., Cottini, A., Umar, A., Klinger, C., Llamas, W., ... Wong, R. (1995). The FinCEN artificial intelligence system: Identifying potential money laundering from reports of large cash transactions. *The 7th Conference on Innovative Applications of AI*.

Senator, T., & Goldberg, H. (1996). Restructuring databases for knowledge discovery by consolidation and link formation. *The Second International Conference on Knowledge Discovery and Data Mining*.

Senator, T., Goldberg, H., & Wong, R. (1998). Restructuring transactional data for link analysis in the FinCEN AI system. *AAAI Technical Report FS-98-01*.

StreamBase. (2012). Powerful real-time architecture for today's high performance modern intelligence systems. *Federal Government, Defense, and Intelligence applications of Year 2012*.

StreamBase. (2013). StreamSQL Guide. Retrieved from  
<http://www.streambase.com/streamsql/>.

Wasserman, S., & Faust, K. (1994). Chapters 1, 2, and 13. In *Social Network Analysis: Methods and Applications*. Cambridge University Press, London.