WWN Software LLC

Blind Encrypted Data Matching — Demonstration Zero

WWN Software LLC pbaker@wwnsoftware.com

Blind Encrypted Data Matching — Demonstration Zero

Table of Contents

Purpose

The software for blind encrypted data matching (BEDM) has been tested in a variety of ways; however, many potential users have mentioned a desire to see the software operating in an exceptionally simple, context-free demonstration that uses generic textual data from the Internet and focuses all the attention on the unique aspects of the BEDM method. The technology used in a simple demonstration can be easily evaluated because there is no need to consider a social, economic, or political context. In this document, we present just such a simple demonstration, which we call Demonstration Zero.

Demonstration zero is context-free meaning that:

- there is no application domain
- there is no representation of geographic, security, or political boundaries in the enterprise data system
- there is no commitment to a specific enterprise architecture
- there are no special software technologies other than the essential and distinctive procedures provided by the BEDM software

The demonstration does require, however, Java Standard Edition 6.0; but that requirement is easily satisfied by a variety of host computer systems.

In this document, we discuss the data, the matching method and the operation of the demonstration. These topic should be accessible to all readers. Then we describe the specific software modules that are tested by the demonstration. The detailed discussion later in this document will be more interesting to software developers and system engineers.

General Plan for Demonstration Zero

BEDM is a method that allows different people or agencies to work on matching data in encrypted form so that each party's security requirements are met. The demonstration needs to show multiple parties. In this demonstration there are four parties playing four roles.

First, there is the person in an administrative role who starts and stops the demonstration and opens various window displays to show the internal operation of BEDM. In a real system, administration of the operation is a more complex job. Also, note that in a real system, no person can inspect all parts of the system from one location. All parts of Demonstration Zero run on one computer with no security wall between the parts. Thus, we compromise the overall security that would be essential for an operational system but we gain the ability to look over all the internal steps during the demonstration.

Moving on to the important functional roles, we have two parties who hold secret data that each must protect. We call these the Blue Team and the Green Team for the purpose of the demonstration.

Finally, the Blue Team and Green Team need an independent agent to match encrypted data for them. That party is called the Blind Agent Negotiator (BAN).

The presumed backstory for the demonstration is that the Blue Team and Green Team work separately, they do not trust each other, and they have not been cooperating. Each possesses documents that may relate to documents held by the other Team. Both teams would benefit if they exchanged related documents but both teams see a security or mission risk if they freely exchange all the documents. Thus, they have a motive to employ the services of the BAN to match encrypted documents and identify those that are related.

In this demonstration, the BAN does not see the actual document text, even in encrypted form. Instead, the BAN must identify related documents by examining encrypted word counts derived from the documents. After a match of related documents has been found, the actual exchange of secure documents must be handled through other channels. It would have been easy for the BAN to receive encrypted documents and provide the matching ones to the Blue and Green Team. We did not configure this demonstration for that option because we wished to show the most conservative, most secure policy for BEDM operation.

Demonstration Data

We use text documents for this demonstration and show the BEDM software in an operation that matches related documents using only encrypted information. Our method for matching is a simple formula based on word counts in the documents. Documents that use similar, distinctive words are regarded as related. Of course, an effort is made to avoid an influence from common words like “a” or “the” which appear in all English documents.

Our documents were obtained from the public domain of old documents that are not subject to any copyright restriction. There were downloaded over the Internet from the site authorama.com. Each document is an essay written by the same author, Francis Bacon, in the 17th century. There are 59 such essays and we have handled them in a way that splits the collected essays into 59 distinct essays for the Green Team and a different set of 59 distinct essays for the Blue Team. The story line for the demonstration is that the Blue Team and the Green Team receive fragmentary information about a subject and the only way either Team can put together all the information and read the full essay is by finding the missing related information in the other Team’s possession.

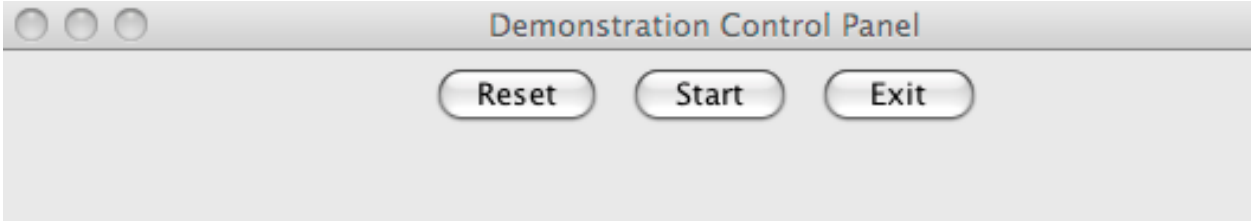
To create this initial distribution of secret data, we took each essay and divided the text into blocks of 50 words. Even numbered blocks went into one new, abridged essay and the odd numbers went into a separate abridged essay. Each team received half the word blocks in the original essay.

While we preprocessed each essay into two distinct abridged essays, we also formated the documents for XML data exchange. Formatting is simple. There is a body containing the text and a title. In addition, we calculated word counts for the text body and placed the word counts in the XML document. At the end of the preprocessing, each Team has XML information for 59 essays and the associated word counts. That information is the starting point for the demonstration.

Demonstration Roles and Display Screens

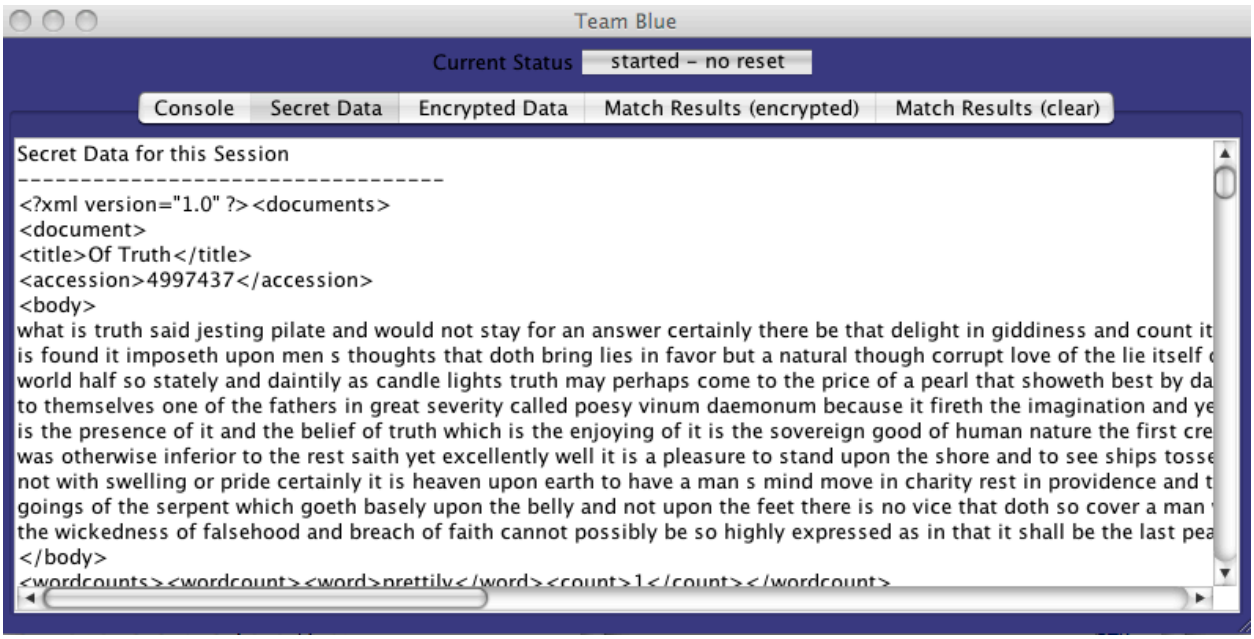
Operations Manager

In the demonstration, the whole operation is supervised by a computer user who runs the demonstration. The software provides a control panel that contains buttons to initiate three operations as shown in the following:



Blue and Green Teams

The two teams manage the data that they have acquired and protect from outside access. For the demonstration, the data holdings are static; although, the holdings would grow in any real-world operation. For each team, there is an independent software object that runs in response to encrypted messages from the other agents in the demonstration. This software object is visible on the demonstration screen as a window that shows the status, the processing activity, and the data documents for the Team. The following picture shows the window for the Blue Team:



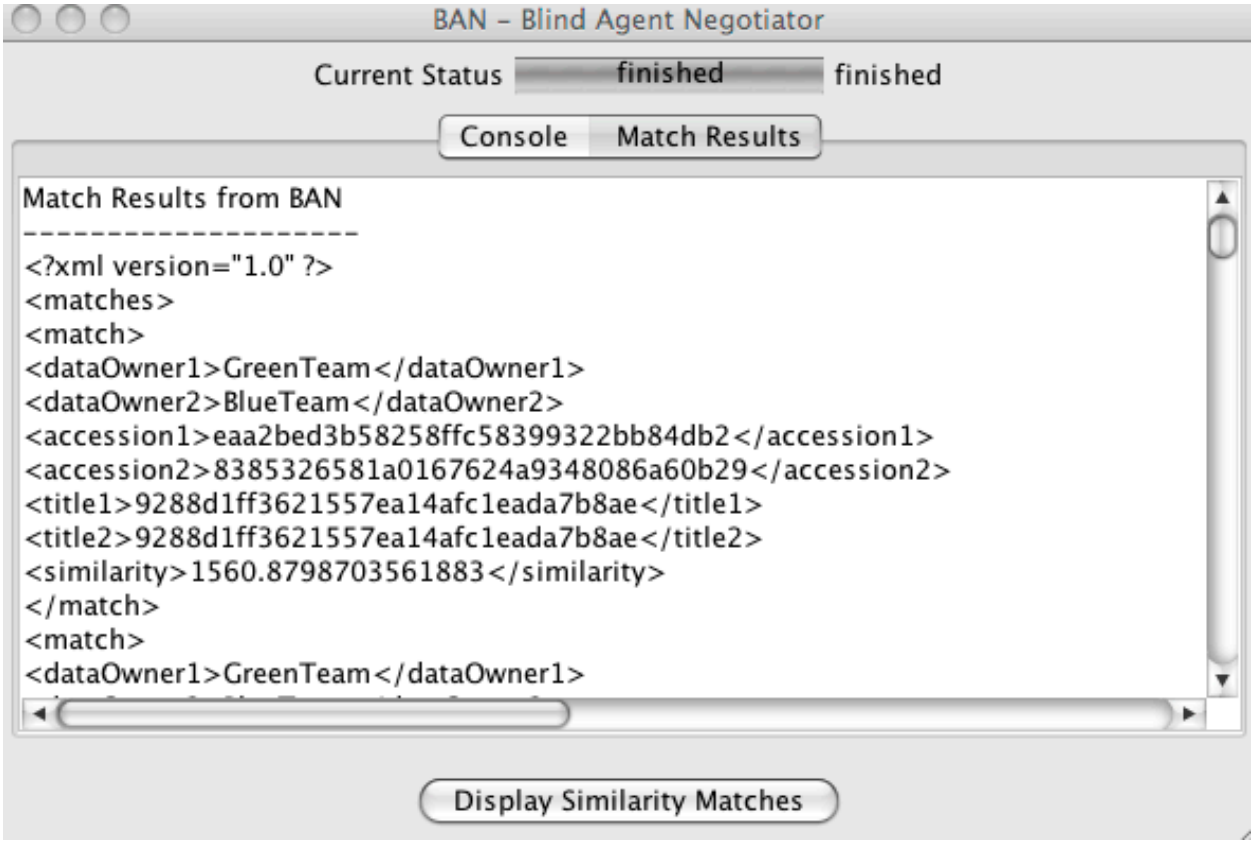
The window for the Green Team is identical except for the obvious changes to the name and color of the panel. Each of these Team windows has a tabbed text area that can show: *Console* (progress messages from the software), *Secret Data* (the secure hidden data that the Team has acquired), *Encrypted Data* (the same secure data after encryption by the session key), *Match Results* (*encrypted*) (the match results from the BAN protected by the session key), and finally *Match Results* (*clear*) (the match results from the BAN in clear text. Note that the window cannot show the data after it has been encrypted by all three keys because such data resides in a block, binary format.

The line above the tab bar is the status and progress line. It will always show the current status of the team during the matching session. While the team's software is actually processing data, the area changes to a progress bar.

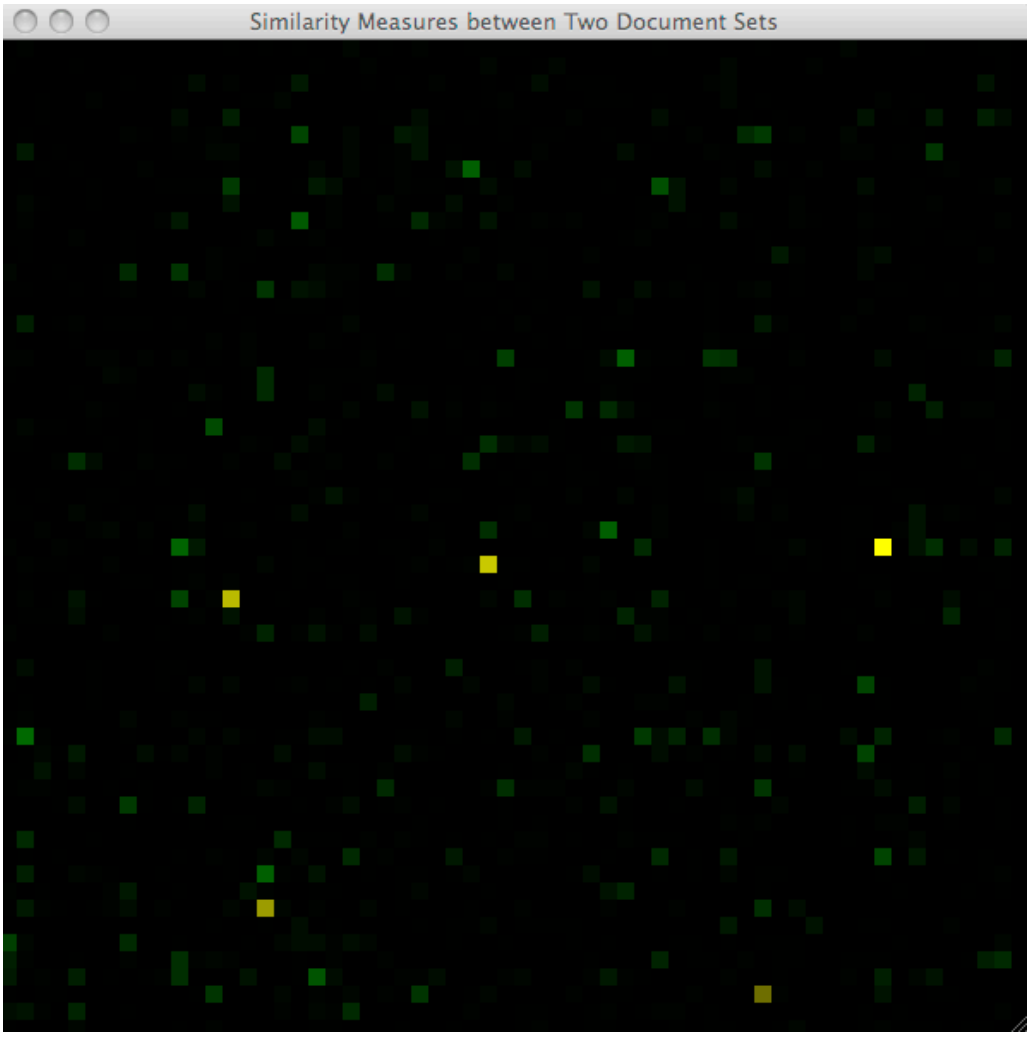
Blind Agent Negotiator - BAN

The BAN plays the role of the middleman in the BEDM process. It is implemented by a software object that runs in response to encrypted event messages and it is visible in the demonstration through its software window. Below, we show a snapshot of the BAN's window taken after the conclusion of the matching process. The text area shows part of the match results to illustrate that the result of the match is written in well-formed XML with encrypted fields protected by the session key.

When the match is finished, a new button appears at the bottom of the window: *Display Similarity Matches*. The button is not visible at other times because the information is transient and it is not available before completion or after a "Reset" or "Exit". The full output of the matching operation is held temporarily as part of the demonstration.



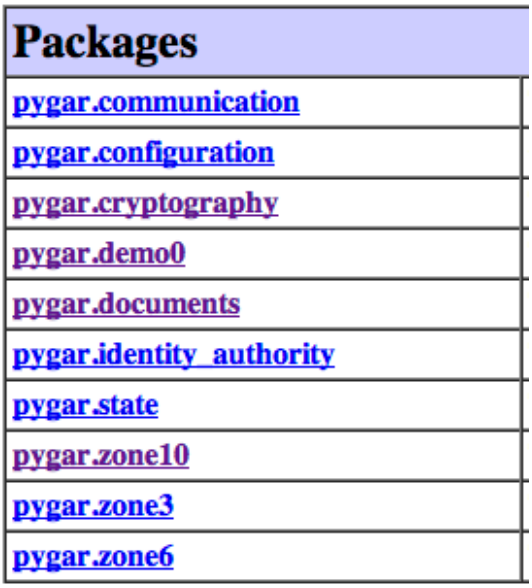
We need to jump ahead for the moment in order to indicate what this button actually does. During the matching, the BAN compares each encrypted document from the Green Team with each encrypted document from the Blue Team. The BAN calculates a similarity index for each pair of documents. The BAN selects the 5 most similar documents and notifies the Blue and Green team about those pairs of similar documents. Meanwhile, the BAN is still holding all the similarity comparisons between documents. When you push the "Display Similarity Matches" on the BAN window, the software uses the temporary comparison data to display the full results. The following window illustrates the display:



The visual display shows a matrix of values where the documents from one team are arranged along the horizontal axis and the documents from the other team are arranged along the vertical. Each color square in the picture represents one comparison. The color depends on the similarity of the two documents. The brighter the green, the stronger the similarity. The computer marks the five most similar documents by shifting the color from green to yellow.

Code Packages Showcased in the Demonstration

The figure to the right shows the hierarchy of Java packages in the software code tree as of May 2010. (Note, there is now a single zone package as of 2012.)



Executable Programs

The actual demonstration is an executable program called *DemoZero* found in the *pygar.demo0* package. The same package also contains a program *PrepareNegoiationFromText* that is used to convert the text document from the Internet into the XML text documents that are the *secret data* for the demonstration.

Cryptography

Another package of interest is the *pygar.cryptography* package. It defines our simplified API (application program interface) that exercises the Java SE 6.0 cryptographic features. It is important to note that SE 6.0 ships with civilian grade cryptographic capabilities and those are the ones used in the demonstration. However, the SE 6.0 software will automatically use the best cryptographic algorithm available on the machine where it is running. That means that government and military applications can use restricted, advanced cryptographic algorithms. However, the Java convention for applying the best algorithm must be replaced in production system. For BEDM to work, all the parties must use the same advance cryptographic algorithm. Thus, the production version will specify which algorithm to use for each matching session. As a result of this policy shift, two agencies may decide not to start a session because one of the parties in the session cannot match the minimum strength of algorithm required by the other parties. It is unlikely that this will cause any practical problem; except perhaps for international operations.

The BEDM encryption process is broken up into software functions that are distributed in *Zones*. A zone is a boundary in the software defined by a package. Thus, this

demonstration has adopted three zones represented in packages: *zone3*, *zone6* and *zone10*. In principle, a production system could use the zones and allocate the zone packages to physical hardware machines in different cyber-security zones. Thus the most sensitive software appears in *zone10* which should be allocated to hardware running behind many firewalls and physical security defenses. The three zone system that was adopted for the demonstration is configured like an onion or set of nested boxes. The outer layer of the onion (*zone3*) is closest to the Internet and contains the least sensitive information. The deepest zone (*zone10*) is the furthest away from intrusion.

It is not our plan to continue the same 3 zone system in the next release of the software. In view of our recent experience with the software and some additional thought on the matter, the future zones will reflect where the various cryptographic keys are used. In the future, the least secure zone will have access only to the public-key identity-certificates. Middle zones will have access in addition to the private-key identity of the local party. The innermost zone will have access to the critical session key that protects the secret data when it moves outside the most secure zone.

State Machines

The package *pygar.state* provides a general utility for defining software behavior through state transition diagrams. The state machine transitions are defined by the software for each role in the demonstration. However, we can give a general description for the state machines here.

State Transitions of the Blue and Green Teams

Each team is capable of preparing data for the sharing operation. Each team responds to a reset command (*Event_Reset*) by the Operations Manager and goes into two parallel wait states: *Wait_Init* and *Wait_Start*. These states wait for events *Event_Init* and *Event_Start*, respectively. Eventually, the teams will wait in a single, non-parallel state for *Event_Match_Finished*.

The sequence following *Event_Init* is:

- *Event_Init*

- Create session key
- Send it securely to all parties, including self
- Send *Event_Start* to all parties, including self, as well as the Coach.

Likewise the sequence following *Event_Start* is:

- *Event_Start*
  - Protect information that should be matched using 3-key encryption process comprising the session key, the Blind Coach's public asymmetric-encryption key, and own private key digital signature.
  - Send protected information as *Event_Data*.
  - Wait *Event_Match_Finished* from the Coach.

Finally, the team receives the results of the match with the *Event_Match_Finished* event:

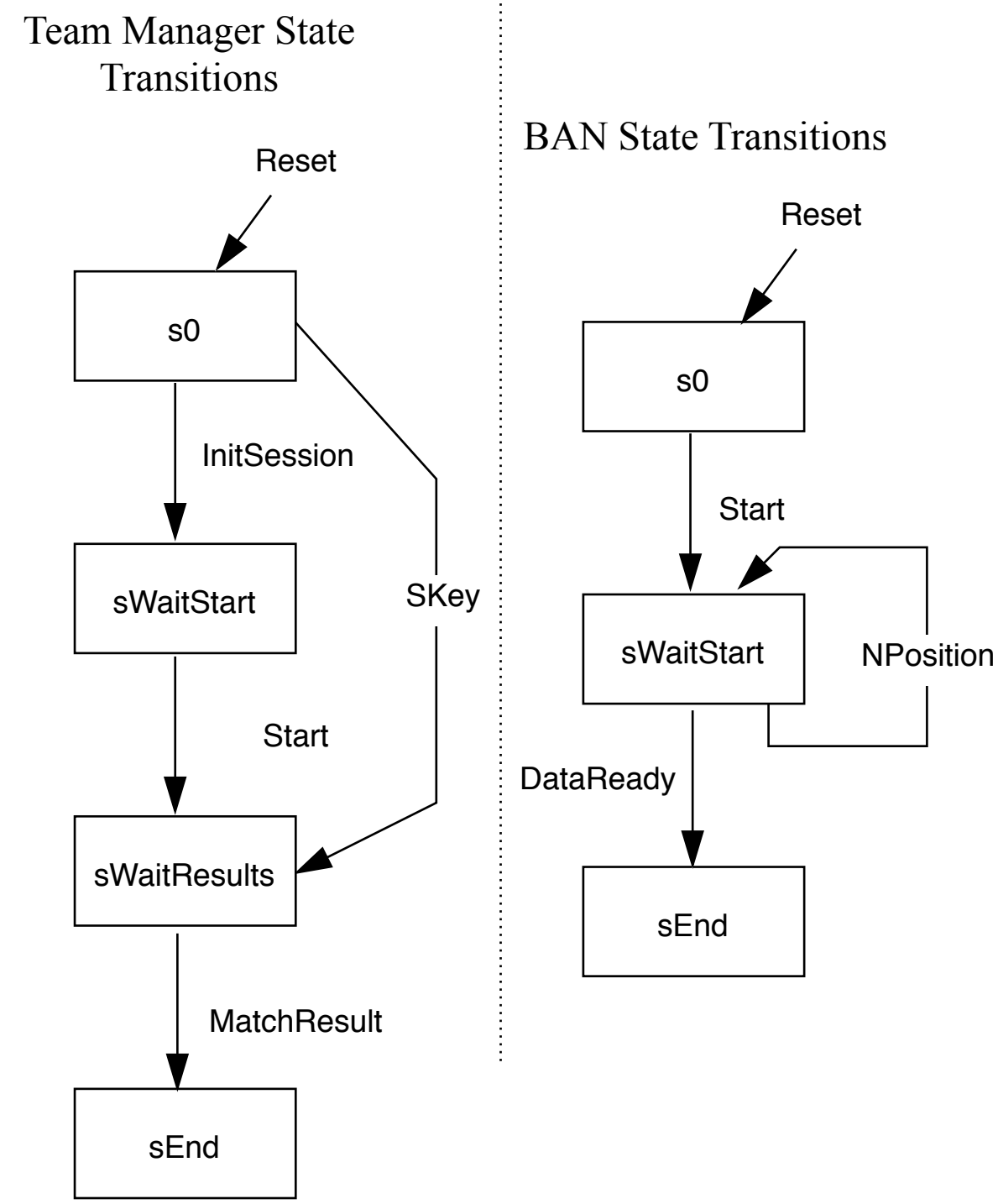
- *Event_Match_Finished*
  - Get encrypted results
  - Extract results from encrypted document
  - Wait for next *Event_Reset*.

State Transitions of the BAN

The BAN responds to Event_Start by waiting for all the data from all the parties:

- *Event_Start*
  - Wait for *Event_Data* from both of the two teams
  - Remove effect of two encryption keys from the documents (session key not available to the Coach)
  - Compare encrypted documents and compile encrypted results
  - Return encrypted results to each team protecting the each document with intended recipients public key and signing it with own private key.
  - Wait for next *Event_Reset*.

Current State Machine Diagrams



Exact Information on State Transitions

The preceding description of the state machine software is reasonably current. However, the software itself is the authoritative source for the state transition diagram of the demonstration. In operation, each team member as well as the BAN operates a state machine that governs its actions. Each of the three entities establishes the state machine at startup using the function:

```
_initStateMachine().
```

A function with this name is found in classes pygar.demo0.TeamMember and pygar.demo0.BAN. Of course, the content of the function is different for a TeamMember and a BAN.

XML Configuration

Because BEDM works with XML based information, it is essential to tell the BEDM software what it should do to protect the XML fields. You can configure the software so that any field that is not especially mentioned in the setup instructions will be removed and never shown during the encryption and matching process.

The XML configuration in Demonstration Zero is written down in the *initFieldTable()* function of the main class *pygar.demo0.ProfileDemo0*. To understand the configuration statements be sure to check the documentation for the *pygar.documents.EncryptedFieldTable* class and the enumeration *EncryptedFieldTable.EFTYPE*. The documentation for the enumeration has a detailed discussion of how the XML fields are handled. To inspect the documentation for Java classes and types, see the instructions in the section "Additional Documentation" below.

Matching Process

The matching is performed in the class *pygar.demo0.CompareFiles*. The match takes the pairs of documents with the best word correlation function *X(d1,d2)* where *X* is the correlation function of document *d1* and document *d2*. The function for *X* is called *xcorrelate* and it is found in the class just cited. It is not a true statistical correlation function but a close relative defined as follows:

- Let *f(w,d)* be the frequency of word *w* in document *d* where word frequency in a document is the number of times the word appears in the document divided by the number of words in the document.
- Let *F(w)* be the ensemble average frequency of word *w* where the ensemble average frequency is the number of times the word appears in all documents divided by the number of words in all documents.
- Let *f'(w,d)* be the normalized frequency of word *w* in document *d* computed as $f'(w,d) = f(w,d)/F(w)$.
- Let *W(d)* be the set of words found in document *d* whilst *W(d1,d2)* is the set of words found in both documents *d1* and *d2* (the set intersection of *W(d1)* and *W(d2)*).
- The correlation X of documents d1 and d2 is X(d1,d2) and is computed as the sum of products $f'(w,d1) * f'(w,d2)$ for all *w* in *W(d1,d2)*.

The matching function will identify documents that are similar based on the occurrence of certain topic words in each. It is a simple function that works sufficiently for demonstration. It is not recommended for practical document analysis because it misses subjects that are uniquely characterized by word phrases rather than single words. Secondly, it is weak in the area of the subject essays by Sir Francis Bacon because the writer is allegorical in many parts and often cleverly ties many thoughts to a single theme diluting the statistical contribution of words from the unifying theme.

Additional Documentation

The software has internal documentation some of which is extracted and formatted for a Java design document (javadoc). This document is formated for viewing in a web browser and the document for demo0 can be viewed at our web site at URL: www.wwnsoftware.com/javadoc.

There are technical matters that pertain to the BEDM software modules that are common not only to this demonstration but also to the production systems. Such matters are described in a separate document entitled: *Design Notes for the Pygar Project*. Note that the *Pygar Project* refers to the software development project launched by WWN Software to build software for BEDM. In addition, see also *Implementation Notes for the Pygar Project*.

Notes

Installation and Operation of the Software

The software is distributed as a Java Archive or JAR file. Installation begins by copying the JAR file to a directory where the software should reside. At this point, the run-time software itself is fully installed because it will run directly from the JAR file. On the other hand, you must provide a set of files and directories for the configuration information and data. At this point, the missing files and directories are inside the JAR file. The next step is to move them outside with the following command:

```
jar -xf demo0.jar demo0
```

The result of this command is that a new directory named "demo0" appears in the file system and contains data and configuration files. For convenience, the data files that are distributed in the JAR file are a snapshot after a demonstration has run. That means that all the results from the previous run are contained in the JAR file and distributed with the demonstration. The idea behind this is that you might want a copy of the results to examine before you erase them and try to repeat the run on a new system.

Additional names could be added at the end as shown here:

```
jar -xf demo0.jar demo0 run runSL src
```

However, you may not want "run" or "runSL" and the "src" may not be present in the JAR file. If we included the Java source code, the "src" symbol extracts a copy of the Java source code and places it in the like named directory.

The "run" and "runSL" files are shell scripts that can start the software on any computer that understands Unix command language. These computers include Unix, Linux, and Macintosh machines. We have tested the software on Macintosh and Windows XP PC's. The latter do not understand shell scripts natively so we consider them separately below.

The shell commands run in a standard command window. On most Unix systems there is a command window named "xterm" and on Macintosh there is the "Terminal"

program in the Utilities folder. The PC command window is called Console and it is found under Accessories.

The "run" command is suitable for a system that has Java SE 6.0 installed but SE 6.0 is not the default. It contains settings that use SE 6.0 for the demonstration. The "runSL" command is a simplified version of the script for systems such as the latest "Snow Leopard" Macintoshes that use SE 6.0 by default. In any case, however, two steps are needed to complete the installation. First, the script must be edited to show the correct path to the data. We recommend using a full path. A relative path causes problems with the current version of the software. Second, enable the script for execution by the following command:

```
chmod a+x run
```

In any case, once the script has been properly prepared, you run the script from the command line as follows:

```
./run
```

On a PC, the scripts do not work unless you have installed an emulator, e.g. the MKS Toolkit. We will assume that this is not the case. Luckily, Java SE 6.0 will likely be the default on your PC. If it is not, you should download and install it from www.java.com. You can check the default version by running the command:

```
java -version
```

If the version says "1.6", then you have SE 6. Finally you can run the demonstration by changing the directory of the console window to the directory where you copied the JAR file. Then type the command:

```
java -cp demo0.jar pygar.demo0.DemoZero datadirpath
```

where *datadirpath* is set equal to the full path of the *demo0* data directory that you extracted from the JAR file earlier.

After a delay, the four screens of the demonstration should appear.

Dedication of the Essays

The documents used to demonstrate encrypted matching were written in the 17th century and have long since passed into the public domain. Nevertheless, authors should have some rights and privileges in perpetuity; therefore, I feel it fitting to copy the dedication of the documents as written by Francis Bacon to afford him a voice in the present day and to acknowledge his source of research funding.

The Right Honorable
My Very Good Lord
the Duke of Buckingham
His Grace, Lord
High Admiral of England
Excellent Lord:

SALOMON saies; A good Name is as a precious oyntment; And I assure my selfe, such wil your Graces Name bee, with Posteritie. For your Fortune, and Merit both, have been Eminent. And you have planted Things, that are like to last. I doe now publish my Essayes; which, of all my other workes, have beene most Currant: For that, as it seemes, they come home, to Mens Businesse, and Bosomes. I have enlarged them, both in Number, and Weight; So that they are indeed a New Worke. I thought it therefore agreeable, to my Affection, and Obligation to your Grace, to prefix your Name before them, both in English, and in Latine. For I doe conceive, that the Latine Volume of them, (being in the Universall Language) may last, as long as Bookes last. My Instauration, I dedicated to the King: My Historie of Henry the Seventh, (which I have now also translated into Latine) and my Portions of Naturall History, to the Prince: And these I dedicate to your Grace; Being of the best Fruits, that by the good Encrease, which God gives to my Pen and Labours, I could yeeld. God leade your Grace by the Hand. Your Graces most Obliged and faithfull Servant,
Fr. Sr. Alban

The modern reader will perhaps note that Bacon proposes an alternative to XML coding to preserve information for future systems.