Pad and Chaff: Secure Approximate String Matching in Private Record Linkage

Aaron D Schroeder
Public Policy Data Group
Institute for Policy and Governance
Virginia Tech
1-540-239-0393
aaron.schroeder@vt.edu

ABSTRACT
The promise of high-quality, integrated data sets for policy analysis is great. However, significant impediments remain that make the creation of such data sets very difficult. The identification of processes and algorithms for carrying out approximate string matching on personal-level information while at the same time keeping that information unknown (private linkage) is quickly growing in importance. Without these processes and algorithms, the creation of high-quality data sets integrated from multiple public agencies remains difficult. In this paper, a method for carrying out secure, private string-matching between data sets using standard string similarity measures is presented.

Categories and Subject Descriptors
H.2.0 [Database Management]: General—Security, integrity, and protection; H.3.4 [Information Storage and Retrieval]: Systems and Software—Performance evaluation (efficiency and effectiveness); H.2.8 [Database Applications]: Data mining.

General Terms
Design, Algorithms, Performance, Experimentation, Security.

Keywords
Record linkage, secure scalar product, private linkage, private information retrieval, linkage, integration, privacy.

1. INTRODUCTION
The integration of record-level data from the administrative data systems of multiple public service agencies has the potential for generating high-quality evidence (heretofore unattainable) to be used in the assessment of public policy effects. However, when attempting to combine data records from these systems, a number of complex legal issues must be considered, not the least of which is the privacy of persons represented by the data in the systems. Significant differences exist between federal law, multiple state/provincial/regional laws, and agency enforcement regulations as they pertain to the integration of identified information [1] [2].

iiWAS2012, 3-5 December, 2012, Bali, Indonesia.

Because there exist many overlapping and often inconsistent privacy-related restrictions at multiple levels of government, the linkage of administrative data records across public agencies can prove exceedingly difficult. In response, an emerging approach to constructing cross-agency data sets for the purpose of policy analysis involves the probabilistic linkage of records using demographics (e.g. name, gender, birthdate, location) that have been hashed/encrypted in some manner so as to be unidentifiable to the entity responsible for carrying out the linking process (or anyone else who may gain unauthorized access to the records).

The issue that arises is that some common methods of demographic comparison (e.g. string similarity functions) are rendered inadequate by the usage of the hashing/encryption techniques. Most notably, a slight misspelling in "first name" will result in completely different hashes and, therefore, a 100% non-match). A number of approaches to rectify this issue have been proposed [3] [4] [5]. However, each proposed method necessarily involves some level of reduction in matching performance as compared to string-similarity functions used on un-obscured data [6].

In this paper, a methodology is presented that provides the necessary function of privacy protection but also allows for the use of existing string-similarity functions (e.g. Jaro-Winkler) on privacy-protected strings without loss of fidelity. The described methodology is currently being deployed in two multi-agency data-integration projects in the State of Virginia [7] [8].

2. Problem Statement
Given two database owners A and B each holding lists of identified strings (e.g. surnames) $S_a = \{a_1, \dots, a_n\}$ and $S_b = \{b_1, \dots, b_n\}$, and a data linker C, compute string similarity measures of all pairs $(a_i, b_i) \in S_a \times S_b$ such that all strings can only be seen by C in a privacy-protected format, and the measures computed match the measures if computed on non-privacy-protected strings.

3. Previous Approaches
A number of approaches have been proposed and used to carry out string similarity matching in a privacy-protected manner. However, given the methods employed, these approaches result in some necessary amount of loss in recall, precision, or both in comparison to string similarity functions carried out on unprotected strings. Pang and Hansen suggest a protocol employing a common table of reference strings to which the actual strings in two data sets can be compared. Edit-distances from the actual strings to the reference strings can be computed. The reference strings closest to the actual strings can be encrypted and sent along with the edit distances to be used for match determination [3]. Unfortunately, testing of the process results in a

fairly sizeable reduction in both recall and precision in comparison to a reference string similarity measure used on unprotected strings [6].

Scannapieco, et al. suggest a protocol that employs a bit of mathematical stenography by embedding a given string using the SparseMap method into a Euclidean space already populated with random strings. The coordinates for a given string are then given as the approximate distances between the string and the random strings. A third party compares the Euclidean distances between the string to determine a match [4]. Testing shows a markedly better result than the table of reference string approach, but still a not insignificant difference from a reference string similarity measure used on unprotected strings [6].

Schnell et al. propose a very novel method that takes advantage of the properties of Bloom filters. A Bloom filter is a space-efficient probabilistic data structure represented by a bit array that is used to test whether an element is a member of a set. Strings are stored as bits that represent the HMAC hashes of the string's constituent n-grams. Bloom filters with similar strings will have a high proportion of the same bits set to 1. Using this knowledge, a string similarity measure, the Dice coefficient, can be used to calculate the ratio of similarity. This method works quite well, in fact testing shows that the precision-recall curves using 2-grams is nearly identical to the benchmark metric [6]. However, the Bloom filter approach is only applicable to using similarity measures that determine the overall similarity of one set of characters to another. That is, when order does not matter. Because of the nature of the encoding scheme and Bloom filters themselves, there is not a way to use this method with standard distance-metrics like Levenstein or Jaro-Winkler. In the approach presented here, both n-gram based similarity functions and distance-based similarity functions can be employed.

4. Method

The algorithm described in this paper is to allow comparison of strings using standard string-similarity functions when the strings must be obfuscated in some secure manner (e.g. hash, encrypt, cipher). Two separate processes are elaborated within. The first, using a *one-time-pad* to create cipher for each string, provides for a type of obfuscation that is both theoretically unbreakable and not vulnerable to frequency analysis [9]. The second process enhances the first by employing the method of chaffing and winnowing which involves the addition to the cipher of fake characters ("chaff") to the valid characters ("wheat") so as to result in all encoded strings being the same length [10].

4.1 Key Creation and Distribution

We start by assuming two parties A and B who want to link data records, and a data linker C to whom parties A and B only want to divulge personal information that has been obfuscated in a fully-secure manner. To initiate the process, data linker C sends to both A and B two keys, a randomly-derived pad-key and a randomly-derived wheat-key. The pad-key is used by both A and B for string ciphering. The wheat-key is used by both A and B for character-position hashing. A and B also, individually, create a chaff-key for the insertion of fake characters and positions before delivering the string back to C. The expectation is that these keys are produced in an automated fashion by a computer program that can handle the randomness requirements of pad-key creation, as well as the production of a globally unique identifier (GUID), a unique 128-bit number that is produced by the computer OS or particular application (e.g. database) or library (e.g. Java, .NET). The wheat- and chaff-keys are based on the production of random GUIDs. The pad-key and wheat-key are sent via SSL-encrypted electronic communications.

After sending the pad-key, the key is immediately discarded from memory (not stored). The wheat-key is retained and stored for chaff-removal and character-position decoding of the returned strings.

4.1.1 One-Time-Pad Key Generation

In cryptography, a one-time pad is a private key that has been generated randomly and is used only once to encrypt a message. The message is then decrypted by the receiver using a matching one-time pad and key. Messages encrypted with keys based on randomness have the advantage that they are considered to be "perfect" encryption methods, which means there is a mathematical proof that cryptanalysis is impossible [9] [11]. Each encryption is unique and bears no relation to the next encryption. There is no pattern to decipher.

4.1.2 Cryptographically Secure Random Key Generation

The key to a one-time-pad's effectiveness is the true randomness of the key produced. Cryptographic applications require truly random sequences that cannot be predicted. This is an issue that demands close attention as most programming languages will instantiate a "System.Random" or similar object instance and call one of the member functions to get random numbers. The numbers returned aren't truly random, but rather pseudo random. This can be good enough for most applications that call for randomness, but not for a one-time-pad because the "pseudo" random nature of the numbers returned by such objects is not good enough for cryptographic purposes. These algorithms generate random numbers that are actually a sequence in which the next number generated is dependent on the previous number generated (therefore potentially predictable).

What is needed instead is for each number to be selected randomly irrespective of any previous selections. Most modern languages implement a library specifically for cryptography and provide such a method for the production of independent random numbers. In Java there is the Security.SecureRandom class. In .NET (which is used for the two Virginia projects) the Cryptography.RandomNumberGenerator abstract class serves as the base class for all cryptographic random number generators. Cryptography.RNGCryptoServiceProvider provides an implementation of that abstract class. For our implementation, we use the RNGCryptoServiceProvider class to create a pad-key by creating a byte array the same size as a given alpha-numeric seed and populating the array with independently-random characters (converted from the numbers).

4.2 Cipher Generation: One-Time-Pad Example

As an example, let's say that we want to encrypt the string "AARON" from data set S_a. Using a cryptographically secure random key generator we feed in the alpha-numeric seed: "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"

and are returned the pad-key

"DO88Z0DLE7SZZI6ABAD4CHIJJ6PTYZUYZYKT".

Table 1. The Message is aligned with the Pad

A	A	R	O	N											
1	1	18	15	14											
D	O	8	8	Z	0	D	L	E	7	S	Z	Z	I	6	A
4	15	35	35	26											

Table 2. Add Message and Pad Positions (from original alphanumeric key), then MOD length of the original key

5	16	53	50	40
Add the two position numbers, then MOD 36				
5	16	17	14	4
Convert Position to Letter				
E	P	Q	N	D

We start by aligning the message and the pad-key and recording each characters position in the original alpha-numeric seed (e.g. R is the 18th letter). We then add the aligned position numbers and used modular arithmetic to "wrap-around" the original alpha-numeric key (in this case, MOD 36) to get to the new cipher-letter. In the example, "AARON" becomes "EPQND". This result is not vulnerable to frequency analysis which can be evidenced by the fact that the same letter is not encrypted the same way twice (The first two letters, "A" and "A" where encrypted as "E" and "P", respectively. The result also withstands index of coincidence attacks (looking for repetition of the key) because the key is as long as or longer than the message being encoded.

While decryption of a one-time pad, given the key, is straight forward, *we have no wish to decrypt the result*. This is why the pad-key is discarded by C immediately after transmission to A and B. What we care about here is that both A and B use the same pad-key, thus producing the same cipher for identical characters. In this manner, we have created an unbreakable cipher that is significantly faster to implement over many thousands of records than any of today's secure one-way hashing algorithms (testing results shown below).

Now let's say we want to encrypt the string "ARRON" from data set S_b.

Table 3. Second Message Conversion

A	R	R	O	N											
1	18	18	15	14											
D	O	8	8	Z	0	D	L	E	7	S	Z	Z	I	6	A
4	18	35	35	26											

The encryption of "ARRON" results in a string with only one character different than the encryption of "AARON". Therefore, string-similarity measures like Jaro-Winkler, Dice Coefficient, and Jaccard-Similarity, etc., *will return the same result as comparing non-encrypted strings*.

4.3 Chaffing the Wheat – An Additional Level of Security

Table 4. Cipher with same Number and Placement of Differences as Non-cipher

5	18	53	50	40
MOD 36				
5	18	17	14	4
Convert Position to Letter				
E	R	Q	N	D

While the use of a one-time-pad results in string encryptions that can withstand frequency and coincidence attacks, there still remains the possible issue of gleaning important information from the actual lengths of the strings themselves. If the attacker knows that a particular column of data store surnames, then some form of statistical attack may be possible just given the incidence of certain name lengths.

For example, while the most common length for a surname in the Unites States is 6 characters (20.82%), a surname of 13 characters is quite rare (0.03%) [12]. Therefore, a 13 character or longer string in the surname column may be quite easily derived (especially given any other demographics, like gender or zip code). It's important to note here, however, that while the deciphering of the 13 character name would certainly constitute a breach, it would not allow for a direct deciphering of other surname encryptions. Recall that the use of a one-time-pad means there is no discernible pattern to the substitution of characters. However, to guarantee that the data records remain de-identified, an additional level of security is surely warranted.

4.3.1 Chaff and Winnow Overview

The second level of obfuscation employed in addition to a one-time-pad is chaffing. Borrowing language from the farming practice of "winnowing the chaff from the wheat", the concept of chaffing and winnowing as a means to achieve confidentiality in message transmission was first proposed by MIT computer scientist Ronald Rivest [10]. Chaffing and Winnowing introduces an approach that does not use encryption keys, but instead uses "authentication" keys. An authentication key allows for the identification of valid bits from invalid bits of data. Using this knowledge, a message can be sent with both valid and invalid parts and the receiver can remove the invalid bits from the good bits (winnow the chaff) to get the message (the wheat). Using this approach, the cipher generated by the one-time-pad can be "padded" with additional characters, resulting in every string returned having the same number of characters.

4.3.1.1 "Wheat" Position Hashes

The wheat-key, as discussed, is a randomly generated GUID sent from party C to parties A and B. The wheat-key is then used by parties A and B to generate *variably-truncated HMAC hashes* as position identifiers for the characters produced by the pad-cipher (the wheat). That is, each position number (1,2,3,etc.) gets hashed by an HMAC using a one-way hash algorithm (e.g. HMAC_SHA512) that uses the wheat-key as the HMAC key. Each position-hash is prepended to its related cipher-character.

The phrase variably-truncated hashes means to indicate that the entire HMAC-hash of a position (e.g. 64 characters for SHA512) does not have to be returned as part of the cipher. It is sufficient that the position-hashes be significantly truncated (e.g. 2 or 3 characters) as long as no position-hash is repeated. In this manner, a significantly shorter string gets delivered saving significant bandwidth (consider a million-row name table). However, to be

able to do so requires that the independently generated chaff-keys produced by parties A and B be generated in such as manner as to check that the generated key, when truncated to the length of the truncated wheat-key position hashes, does not duplicate any of wheat-key position hashes.

4.3.1.1.1 "Chaff" Position Hashes

As indicated, the generation of chaff means the generation of variably-truncated hashes as position identifiers that *do not match* any of the "wheat" position hashes. The random chaff-keys produced independently by both A and B are used for this purpose. These invalid position-hashes are appended to a random selection of enough characters to create a string of some predetermined length (e.g. 20 characters). Because A and B select their chaff keys independently from each other, *the chaff each creates and inserts does not match the other*, thus further obfuscating the string of data.

4.3.2 Fisher-Yates Shuffle and Generation of Return Value

After appending the chaff, an unbiased shuffling algorithm is applied to sufficiently mix up the wheat and the chaff. In our application, the shuffling algorithm employed is a modern variant of the Fisher-Yates shuffle as updated by Donald E. Knuth [13].

The Fisher–Yates shuffle is an algorithm for generating a random permutation (shuffle) of a finite set. Properly implemented, the Fisher–Yates shuffle is unbiased, making every permutation equally probable. The variant employed in our Virginia systems is an "in-place" shuffle, meaning that, given a pre-initialized array, it shuffles the elements of the array in place, rather than producing a shuffled copy of the array. This can result in a performance gain for large arrays.

A summary of the algorithm [14]: To initialize an array a of n elements to a randomly shuffled copy of source, both 0-based:

a[0] ← source[0]

for i from 1 to n − 1 do

  j ← random integer with 0 ≤ j ≤ i

  a[i] ← a[j]

  a[j] ← source[i]

For our purposes, this general algorithm is adjusted to allow for what we call a "shuffle-chunck" size. The shuffle-chunck size equals a single character plus its appended position-hash length. Each "chunk" needs to be shuffled together as a single entity.

4.4 Winnowing the Chaff and Comparing the Wheat - Approximate String Matching with Cipher and Chaff Strings

Upon receipt of the data, the fully chaffed strings are stored for the duration of the linkage procedure. Party C feeds the retained "wheat" key to a pre-compiled comparison-program, along with the two lists of fully-chaffed strings and the desired string-similarity function. Using the "wheat" key as the HMAC key, the comparison-program first generates hashes for the maximum number of character positions (e.g. 1 thru 20). Then, for each possible comparison between strings in S_a and S_b, these valid hashes are used to *pull out* and *re-order* the valid cipher-characters. At this point, the comparison-program compares the two cipher-strings using the selected string-similarity function. The comparison-program returns to party C only the unique-identifiers associated with the chaffed-strings (supplied by both parties A and B), and the string-comparison metric.

4.5 Sample Output

Figure 1 shows sample output from the use of the Pad and Chaff algorithms employed in the data record linkage system of the two Virginia data projects.

After showing the generated wheat-key, chaff-key, and pad-key (here called "Encrypt-safe rand alphanum key"), the sample output shows in a step-by-step manner how the original string to be encoded is (1) normalized (removing odd characters, spacing, etc.), (2) ciphered using the randomly generated one-time pad, (3) extended with the insertion of wheat-key hashed positions, (4) extended with the insertion of fake chaff-key hashed positions, and (5) shuffled using the Fisher-Yates shuffle algorithm. It is easy to see that the final string values that are delivered for matching do not in any way match.

Various string similarity functions are then run by the comparison program on the two final string, producing the metrics at the bottom of the figure.

4.6 Performance Metrics To Be Considered

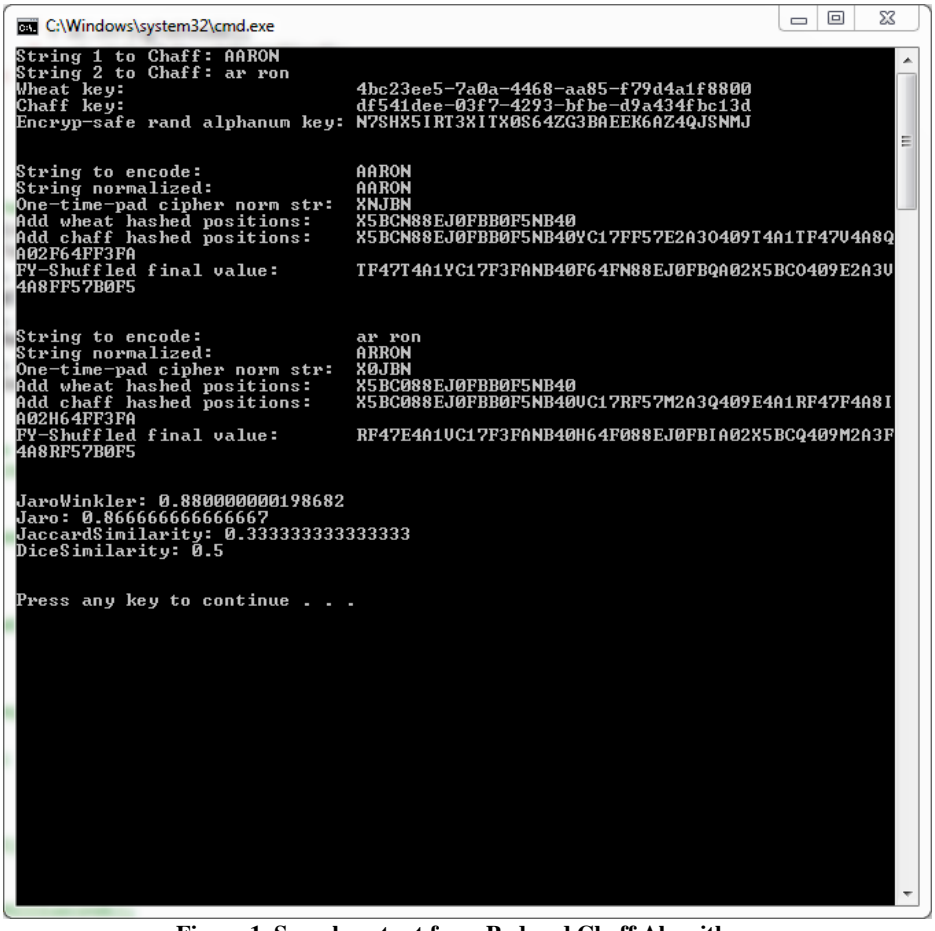

Figure 1. Sample output from Pad and Chaff Algorithms

Because the ciphers that get compared at the end of the process using string similarity functions will produce results identical to those produced by using the same similarity functions, the typical precision and recall metrics need not be considered. The two important metrics to be considered in this process are (1) the speed of the pad-ciphering and chaffing-processes in comparison to other processes that may be used (e.g. simple one-way hash algorithm, using a one-way hash-algorithm to generate the encoded string letters instead of using a one-time pad cipher, etc.), and (2), the size of the data set to be returned in comparison to the same data set being returned using alternate methods (e.g. returning full single one-way hashes of each string). There are many alternate scenarios that can be considered. Preliminary tests indicate that a one-time pad will produce its cipher about 60 times faster than a standard one-way-hash (SHA1), however, there a number of other variables to consider before drawing any conclusions. Work is currently underway to compare a number of alternate scenarios. These will be reported in a follow-up extended version of this document.

4.7 Conclusion

This paper has presented a method for conducting privacy-protected record linkage in a manner that still allows for the use of standard string similarity measures. The integration of record-level data from the administrative data systems of multiple public service agencies has the potential for producing analyses of public policy effects that have heretofore been impossible given the various and overlapping privacy laws and regulations at multiple levels of government that preclude the integration of identified data sets. The hope is that this method will enhance the ability to create high-quality integrated data sets while still keeping personally identifying data private.

5. Works Cited

1 Schroeder, Aaron. *Multi-Agency Integration of Child-Relevant Data Sets in the Commonwealth of Virginia: Application of a Privacy Protecting Federated Model*. The Data Quality Campaign (DQC), Washington D.C., 2009.

2 Spears, J. V., Bradburn, I., Schroeder, A., Tester, D., and & Forry, N. New data on child care subsidy programs. *Policy and Practice* (August 2012), 18-21.

3 Pang, Chaoyi and Hansen, David. Improved Record Linkage for Encrypted Identifying Data. In *Proceedings of the 14th Annual Health Informatics Conference* (2006), 164-168.

4 Scannapieco, M., Figotin, I., Bertino, E, and Elmagarmid, A.K. Privacy Preserving Schema and Data Matching. In *Proceedings of the ACM SIGMOD International Conference on Management and Data* (2007), 653-654.

5 Schnell, Rainer, Bachteler, Tobias, and Reiher, Jorg. Privacy-preserving record linkage using Bloom filters. *BMC Medical Informatics and Decision Making* (2009).

6 Bachteler, Tobias, Schnell, Rainer, and Reiher, Jorg. An Empirical Comparison of Approaches to Approximate String Matching in Private Record Linkage. In *Proceedings of Statistics Canada Symposium -- Social Statistics: The Interplay Among Censuses, Surveys and Administrative Data* (2010).

7 Education, Virginia Department of. *Virginia Longitudinal Data System (VLDS)*. http://www.doe.virginia.gov/info_management/longitudinal_data_system/index.shtml, Richmond, VA, 2012.

8 Social Services, Virginia Department of. *Project Child HANDS: Child Care Subsidy, Health and Early Education*. http://www.childhands.org/, Richmond, VA, 2012.

9 Denning, Dorothy. *Cryptography and data security*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1982.

10 Rivest, Ronald L. *Chaffing and Winnowing: Confidentiality without Encryption*. MIT Lab for Computer Science, Boston, MA, 1998.

11 Claude, Shannon. Communication Theory of Secrecy Systems. *Bell System Technical Journal*, 28, 4 (1949), 656-715.

12 Word, David L, Coleman, Charles D., Nunziata, Robert, and Kominski, Robert. *Demographic Aspects of Surnames from Census 2000*. United States Census Bureau, Washington D.C., 2000.

13 Knuth, Donald E. *The Art of Computer Programming vol. 2 (3rd ed.)*. Addison-Wesley, Boston, 1969.

14 *Fisher–Yates shuffle*. Wikipedia, 2012.

15 Culhane, Dennis P., Fantuzzo, John, Rouse, Heather L., Tam, Vicki, and Lukens, Jonathan. Connecting the Dots: The Promise of Integrated. Data Systems for Policy Analysis and Systems Reform. In *Intelligence for Social Policy* (), University of Pennsylvania.