

Privacy Preserving Electronic Surveillance *

Keith B. Frikken
CERIAS and Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
kbf@cs.purdue.edu

Mikhail J. Atallah
CERIAS and Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
mja@cs.purdue.edu

ABSTRACT

Can protocols make privacy concerns no longer clash with security imperatives, by satisfying both? The former seems to preclude the widespread collection and sharing of data about individuals and their activities, whereas the latter (especially national security and law enforcement) seems to require it. This paper gives a step in the direction of satisfying both, by giving protocols that make the data-sharing about individuals and their actions *conditional on these individuals being already on a list of known suspects, deadbeats, criminals, etc.* More formally, if we call U the set of all identities, S the subset of U for which monitoring is authorized, Alice the monitoring agency (that alone knows S), Bob any of the data-collection entities, $p \in U$ the identity whose activity Bob has just observed, then the outcome of the protocol is that Alice learns the activity of p if and only if $p \in S$, and Bob does not learn anything about the membership of p in S .

Categories and Subject Descriptors

H.3.m [Information Search and Retrieval]: Miscellaneous

General Terms

Security

Keywords

Surveillance, Privacy, Security Protocols

*Portions of this work were supported by Grants EIA-9903545, IIS-0219560, IIS-0312357, and IIS-0242421 from the National Science Foundation, Contract N00014-02-1-0364 from the Office of Naval Research, by sponsors of the Center for Education and Research in Information Assurance and Security, and by Purdue Discovery Park's enterprise Center.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WPES'03, October 30, 2003, Washington, DC, USA.
Copyright 2003 ACM 1-58113-776-1/03/0010 ...\$5.00.

1. INTRODUCTION

The ability to monitor suspects electronically is useful to create electronic dragnets for suspected terrorists, criminals, dead-beats, etc. A type of electronic surveillance is to bind certain activities (credit card purchases, patterns of travel, purchase of certain goods like firearms or certain chemicals, etc) to identities. This type of surveillance allows a monitoring agency to know when an identity is used, which would allow this agency to locate suspects or to monitor suspects' activities to evaluate their threat level. However, such a system has severe privacy concerns; this monitoring system should *not* be usable to track people whose surveillance is not authorized, nor should it leak out the identities of those under surveillance.

Privacy advocates have been challenging attempts to bring more and more information into integrated collections and centralized repositories, where algorithms can be run against the data. The national-security-motivated "Total Information Awareness" program has come under attack (e.g., U.S. Senate Bill 188, the Data-Mining Moratorium Act of 2003) [6]. The issue here is that *all* of the citizenry is subjected to an unusual level of scrutiny, to protect society from a rather small set of individuals. It is desirable to "target" this scrutiny to the set of suspects rather than subject everyone to it. One problem with confining the scrutiny to the list of potential suspects is that this appears to require publicizing the list, widely disseminating it, keeping it up to date and its multiple copies consistent, etc. Not only would this pose unusual technical difficulties, but it also poses serious legal and ethical ones as well (e.g., the list does not exclusively consist of convicted criminals). There are many reasons for the necessity to hide the list of suspects from the entities that collect identity-activity information (travel industry, financial industry, etc), for example:

- To protect the data-collecting entities from potentially dangerous individuals (consider a suspected killer using a credit card).
- To protect the rights of the suspects.
- To prevent the data-collecting entities from unauthorized disclosure of the fact that these individuals are suspect. This could jeopardize a law enforcement investigation, tip off the suspects that they are under surveillance, damage the reputations of innocent people, etc.

Privacy-preserving electronic surveillance technology has the potential to advantageously resolve the apparently con-

tradictory concerns of privacy advocates and of national security and law enforcement imperatives, e.g., making possible “privacy-preserving information awareness”. This paper gives a step in this direction, by giving protocols that make the data-sharing about individuals and their actions *conditional on these individuals being already on a list of known suspects, deadbeats, criminals, etc.* More formally, let U be the set of all identities, S be the set of suspects for which monitoring is authorized (note $S \subseteq U$), Alice the monitoring agency (that alone knows S), Bob any of the data-collection entities, Polly (a possible suspect) with identity $p \in U$ whose activity Bob has observed. The outcome of the protocol is that Alice learns the activity of p if and only if $p \in S$, and Bob does not learn anything about the membership of p in S . If John Doe, who is not in the monitored list S , buys a large quantity of fertilizers, then the monitoring agency Alice learns that *someone* has bought a large quantity of fertilizers but does not know it is p who did so; she only learns the identity p , if $p \in S$. This is made stronger in Section 6, where all Alice learns is that someone had a transaction with Bob.

This system does not completely solve the problem of private electronic surveillance. The assumption that the set of suspects is already known to a monitoring agency is limiting, because it does not support the detection of unknown suspects based on their observed behavior. However, this is a first step in the direction of private electronic surveillance. If this first step is not possible, then there is little hope that a more complicated task is possible. Furthermore we assume the operation to be a real time transaction by transaction system, which is the most responsive architecture, but it may not be the most scalable system. However, the protocols in this paper can be modified to fit other types of architecture. For example, if Bob stores the events at his site, this system can be modified to do post-event analysis. This has two primary applications: i) after a major incident the authorities have limited the set of suspects to a small group, this surveillance system could be used to further reduce the set of suspects, and ii) after an event has occurred where the authorities know the culprits, information on typical behaviors before events would be useful to detect such events in the future. Also, the techniques in this paper can be used in a batch fashion, i.e., Bob collects a large number of events, and then submits them to Alice in a large batch.

2. RELATED WORK

There is a huge literature on how to achieve anonymity in computer systems and networks, in payment systems, and in e-commerce. We will refrain from going into these because we are mainly concerned about the identity-based kinds of commercial, manufacturing, and financial interactions that underpin much of our society and economic system: In these, anonymity is often impossible due to the legal or commercial necessities of knowing who is one’s counterpart in any given transaction; these necessities are dictated by the law, by accounting rules, by liability considerations, etc. Even when cash or e-cash payments are used (e.g., for an airline ticket, physical goods bought online, etc), other considerations often force the disclosure of more information (a name must be attached to an airline ticket, a shipping address to an online purchase, etc).

On the problem of privacy-preserving monitoring of a computer system for certain patterns of misbehavior, the paper

[1] gave a pseudonym-based design such that a threshold of misbehavior by a pseudonym automatically reveals the true identity hiding behind the pseudonym. For example, if pseudonym p ’s event sequence matches k or more times various attack patterns (stored by the monitoring system in a database of such attack patterns) then the veil is lifted on p ’s identity. In the clever scheme of [1], the event sequences of pseudonyms who do not misbehave are not tied to their true identities, only to their pseudonym. This scheme was designed for audit trails of computer systems, not for a national pseudonym-based system for economic and social transactions (and it is doubtful that it would work in that broader context).

The Private Information Retrieval (PIR for short) problem considered in the literature consists of a client and server; the client needs to get the i th bit of a binary sequence from the server without letting the server know the i ; the server does not want the client to know the binary sequence either. A solution for this problem is not difficult; however an efficient solution, in particular a solution with small communication cost, is not easy. Studies [5, 4, 14, 7, 17, 2, 10, 9] have shown that one can design a protocol to solve the PIR problem with much better communication complexity than by using the general theoretical solutions from the area of secure multiparty computing. Our motivation is also to come up with efficient solutions, but in our problem the roles of who should know what are reversed.

3. BUILDING BLOCKS

In this section we look at the computational building blocks necessary for our protocols. This section is organized as follows: in Section 3.1 we briefly review commutative encryption and oblivious transfer, in Section 3.2 we review the area of Secure Multiparty computation, in Sections 3.3-3.5 we introduce various protocols that are used in our scheme (note that none of these protocols is novel), and in Section 3.6 we give a brief summary of these protocols.

3.1 Commutative Encryption/Oblivious Transfer

Given two commutative encryption functions E and F the following property holds: $E(F(x)) = F(E(x))$. Commutative encryption can be used to implement Oblivious Transfer (OT). There are many equivalent definitions of Oblivious Transfer. In this paper we use the definition of 1-out-of- k OT to be that Alice has a set of items x_1, \dots, x_k and Bob has an index $i \in \{1, \dots, k\}$. The OT protocol allows Bob to obtain x_i without revealing any information about i to Alice and without revealing any information about other x values to Bob. For more information on commutative encryption and OT the reader is referred to [18]

3.2 Secure Multiparty Computation

Secure Multiparty computation was introduced in [19], which contained a scheme for secure comparison; suppose Alice (with input a) and Bob (with input b) desire to determine whether or not $a < b$ and leaking nothing else besides this result. More generally, SMC allows Alice and Bob with respective inputs a and b to compute a function $f(a, b)$ in a private manner for some function f . If there is a trusted third party Trent then any such protocol is trivial since Alice and Bob could reveal a and b to Trent, and then Trent could compute $f(a, b)$ and reveal the result in the appropri-

ate manner; however, there is no such trusted third party in many cases. A protocol is said to compute a function f privately if it reveals nothing to a given party other than what can be deduced from that party’s input and their output alone. Elegant general schemes are given in [12, 11] for computing any function f privately. However, these general solutions are considered impractical for many problems, and it was suggested in [13] that more efficient domain-specific solutions can be developed.

A further note that needs to be addressed about SMC is what is meant by “learning a result”. A result can be learned in several ways including: i) it can be asymmetric (i.e., one party learns the result), ii) it can be simultaneous (i.e., both parties learn the result), or iii) it can be split in some fashion (i.e., neither party knows the result but they each have a share and when these shares are combined they produce the result). An example of a split result is that the data could be split in an XOR-wise fashion. Split knowledge has two interesting properties: i) the other forms of knowledge reduce to it with a single step (asymmetry can be achieved by having one party send his share to the other party, and simultaneity can be achieved by using a fair exchange [18] of the shares) and ii) it can be used to compute intermediate computations which can be used to compute result without revealing these intermediate values.

3.3 Secure Asymmetric Equality

In this section we review a well-known protocol for asymmetric equality checking. The protocol listed here uses the well known “double encryption” trick.

Input: Alice has input x and Bob has input y . Assume $x, y \in \{0, \dots, N - 1\}$.

Output: Alice knows whether or not $x = y$.

1. Alice chooses a commutative encryption function (call it E_A) and sends Bob $E_A(x)$.
2. Bob chooses a commutative encryption function (call it E_B) and sends Alice $E_B(y)$ and $E_B(E_A(x))$.
3. Alice computes $E_A(E_B(y)) = E_B(E_A(y))$. She then compares this to $E_B(E_A(x))$, from which she can determine if $x = y$.

The above protocol requires 2 communication rounds and $O(\log N)$ communication.

3.4 Secure Split Comparisons

For two k -bit numbers ($x = x_1 \dots x_k$) and ($y = y_1 \dots y_k$), there are circuits exist for equality testing ($(x = y) \equiv \bigwedge_{i=1}^k (x_i = y_i)$) and comparison ($(x < y) \equiv \bigvee_{i=1}^k ((\bigwedge_{j=1}^{i-1} (x_j = y_j)) \wedge (y_i \wedge \neg x_i))$) whose number of gates is linear in the number of bits, and whose depth is logarithmic in the number of bits. Thus using circuit simulation [12] it is possible to do a split equality check or a split comparison with $O(\log N)$ communication and $O(\log \log N)$ rounds (recall that, in the context of surveillance, N is the size of the population).

3.5 Secure Selection

This protocol allows Alice and Bob to obliviously select a value from a list without knowing which value is selected, furthermore the result is split between them. This is useful in that it allows them to use previous computations to

select a value from a set that can then be used in other situations. For example to find the maximum of two elements, one could do a split comparison and then use this protocol to select the larger of the two. This protocol is useful in many SMC computations, for example see [16]. Next we describe a protocol for doing this when there are two items, but this can be extended to an arbitrary number of items.

Input: Alice has input bit r and Bob has input bit r' and the pair of items k_0, k_1 that are the transferable items for the OT.

Output: Alice and Bob have a split value x and x' respectively, where $x \oplus x' = k_{r \oplus r'}$.

1. Bob chooses a hiding factor h and creates a table T , where $T = \{h \oplus k_0, h \oplus k_1\}$ if $r' = 0$ and $T = \{h \oplus k_1, h \oplus k_0\}$ if $r' = 1$.
2. Alice and Bob engage in a 1-out-of-2 OT protocol where Alice retrieves the r th entry of T . Bob’s outputs is h and Alice’s output is the entry she retrieves. Note that Alice’s output will be $h \oplus k_{r \oplus r'}$.

3.6 Summary of building blocks

Here we give a summary of the building blocks we use in this paper so that we can conveniently refer to them later.

1. *Asymmetric Equality:* Alice and Bob engage in a protocol so that Alice learns if $x = y$. (Section 3.3)
2. *Secure Comparison:* Alice and Bob engage where they learn (split or asymmetric) whether or not Alice’s item is less than Bob’s (the comparison can be greater than, less than or equal to, equal to, etc.) in a split fashion. (Section 3.4)
3. *Equality with Selection:* Alice has input bit e_1 and Bob has input bit e_2 and a pair of items x_1, x_2 . Alice learns x_1 if $e_1 = e_2$ and learns x_2 otherwise. (Section 3.5)

4. PROTOCOLS

Suppose U is the set of all identities (assume $|U| = N$), and Alice has a subset S of U , $S = \{s_1, \dots, s_n\}$. S consists of the suspects that Alice wishes to monitor. Let Bob be an entity that collects “identities” (really identity-activity information, but we use the term identities as a shorthand) and is willing to provide Alice with the information she needs to achieve the monitoring. Let p represent an identity that Bob has collected. Bob and Alice wish to engage in a protocol so that Alice learns p if and only if $p \in S$. It is assumed that $N \gg n$, but it is also assumed that it is tractable to perform $O(N)$ work. The second assumption mimics the real world, where N would be the population of the monitored group (in the United States, N is about 270 million according to 2000 census data [3]). The rest of this section is organized as follows, each subsection contains a new scheme for achieving private electronic surveillance. The schemes are different in communication efficiency (rounds and bandwidth), levels of privacy, and scalability. Many of the schemes build on each other, with each subsequent scheme enhancing previous schemes. The reader that is only interested in our strongest schemes should skip to Sections 4.7 and 4.8 to see our strongest protocols. A brief summary of the schemes is given in section 4.9.

The protocols in this paper assume that the players (Alice and Bob) are “honest but curious”. Recall that, in the honest-but-curious model, the parties follow the protocols’ steps (honesty) but will try to compute anything they can with the information they learn through the protocol (curiosity). This is an appropriate model for Bob, for if he wished to sabotage any protocol he could simply modify his inputs, so we assume that Bob voluntarily helps Alice. However, it is problematic to assume that Alice is honest but curious, since she controls the set S . What would happen if Alice lets $S = U$? In this case she would be able to monitor every person in the population, which appears to negate our protocols. However, in our privacy preserving protocols Bob will learn the size of S (or at least a close estimate of it) and our stronger protocols require Bob to perform $O(|S|)$ encryptions as a setup cost. This mitigates Alice’s “ $S = U$ ” attack because although it is tractable to do $O(N)$ encryptions it would be unbearable for many Bob’s; furthermore Bob’s knowledge of the size of S will allow Bob to know if Alice is using a particularly large S . However, this does not prevent Alice from adding a few additional suspects to S and monitoring people which she has no authority to monitor; this problem (of preventing Alice from arbitrarily changing S in an unauthorized manner) is addressed in Section 5.

4.1 Protocol 1: No privacy

This scheme is just a warm-up that does not protect the privacy of non-suspects (i.e people not in S). When privacy is not a concern, Bob can send Alice the identity p , and then Alice will be able to determine if $p \in S$. However, doing this allows Alice to perform surveillance for any entity in U (not just those in S). It is obvious that Bob learns nothing in this protocol other than what he already knows. Furthermore, this scheme only requires 1 communication round and $O(\log N)$ communication and there is no communication required for Alice to update S . This scheme minimizes communication but also minimizes privacy.

4.2 Protocol 2: Maximizing Privacy

In this section we look to provide complete privacy without regards to communication costs (which we reduce later). Alice and Bob engage in the following protocol:

1. Alice and Bob engage in n *Asymmetric Equality* protocols, the i th of which is made between s_i and p .
2. If the i th comparison is true, then Alice knows that $p = s_i$, and hence knows p .

The above scheme requires $O(1)$ communication rounds but requires $O(n \log N)$ communication; also, updates require no communication. Furthermore, its privacy is as strong as the *Asymmetric Equality* protocol.

4.3 Protocol 3: Trading privacy for communication

The above scheme can be made more communication efficient by using the same E_A and E_B (for the *Asymmetric Equality* protocols, see section 3.3) for many transactions. By doing this Alice does not need to send her entire list to Bob each time, but rather has to send it only once for a set of transactions. In this case there is $O(n \log N)$ communication setup cost, but it requires only $O(\log N)$ communication per transaction. Updates to Alice’s list take only $O(\log N)$

communication for an insertion and require no communication for a removal. However, there are a few problems with this scheme:

1. *Scalability*: The system requires Alice to keep an encrypted copy of Bob’s list for each Bob, which limits its scalability.
2. *Profiling*: Alice will learn if the same customer Polly visits Bob’s business twice. Because Alice does not know it is Polly doing this (assuming Polly is not in S), the privacy impact of this may not be devastating, but clearly a solution without profiling is preferred.
3. *Known plaintext attacks*: Suppose Alice can break Bob’s encryption scheme but has to use a lot of resources to do so. In the previous scheme, Alice learns only a single value if the key is broken, but in this case she could learn all information at Bob’s site by breaking a single key. It is also worth noting that Alice has more information to break the key in this scheme.
4. *Back tracing for updates*: If Alice updates her list then she can perform surveillance in the past about that member. This can be fixed by requiring the encryptions to be redone at each update, but this is expensive.

Problems 2-4 all have the same root problem: Alice sees the information in the same hiding mechanisms, which allows her to use previous information. Thus later schemes inject new randomness into the system each time.

4.4 Protocol 4: Fixing the previous problems

The following achieves similar performance to the scheme outlined in Section 4.2, but it is another method of doing it which should be viewed as a transition to the next class of schemes. The key difference here is to prevent Alice from learning any intermediate information even in encrypted form by using split computations. To prevent profiling it is pertinent that Alice does not learn a value that can be compared with previous values; this is achieved by using *Secure Comparison*. Here we use split resumes instead of the asymmetric protocol, but otherwise it is identical to the protocol in 4.2. At the end of the protocol an additional step of data revealing is done, so that Alice learns the results of the comparisons. This protocol requires $O(1)$ comparison rounds and $O(n)$ comparisons; with the most communication efficient split equality checks this requires $O(\log \log N)$ rounds and $O(n \log N)$ communication. Updates to S require no communication. It is trivial to prove that this is as secure as the split equality check. The remaining sections explore how to reduce the communication cost of such protocols.

4.5 Protocol 5: Reducing the Communication Complexity

In this section we explore using a binary search to trade bandwidth for rounds. This protocol also has reduced privacy over the previous scheme. We use a binary tree in this Section (and subsequent sections), but the tree can be flattened by increasing the degree of the tree. There are two parts to this protocol: i) constructing a binary tree and ii) searching the tree. To construct the tree structure Alice creates a tree with $\lceil \log n \rceil + 1$ levels with the leaf nodes

corresponding to the elements of S . The internal nodes are chosen to be values that form a valid binary search tree. At each level Alice performs a *Secure Comparison* with Bob (who uses p as his input) and they navigate to a leaf node. Upon reaching the leaf node, Alice knows that if $p \in S$, then it must be a specific value. Thus her and Bob can engage in a *Asymmetric Equality* protocol, which allows her to know p if $p \in S$.

This requires $O(\log n)$ comparison rounds and $O(\log n)$ comparisons. Thus this scheme requires $O(\log n \log \log N)$ rounds and $O(\log n \log N)$ communication. Furthermore, updates to S can be made without communication.

The privacy of the above scheme can be described as follows:

1. Bob learns nothing in the above protocol, as long as the *Secure Comparison* and *Asymmetric Equality* are secure. All he learns is the number of comparisons, but this reveals only an estimate of the size of S .
2. If $p \in S$, then Alice learns only the required information. If $p \in S$ and Alice knows that what p is, then she could simulate each comparison (i.e., knowing p is in the set and the value of p reveals the outcomes of the comparisons).
3. If $p \notin S$, then Alice learns some superfluous information. Namely she can put an upper and lower bound on the value of p . This range may not be that wide since the values are structured.

4.6 Protocol 6: Decreasing the likelihood of small ranges

The fundamental problem with the above protocol is that for a given possible suspect Polly, p , Alice can link p to a group of people smaller than U . This is particularly undesirable when this group is small. This can be mitigated by encrypting the data before running the previous protocol. The protocol is as follows:

1. Bob chooses an equality-preserving encryption function E , computes $E(p)$ for himself and sends Alice E .
2. Alice computes $E(s_1), \dots, E(s_n)$.
3. Alice and Bob run the previous protocol with Bob's input as $E(p)$ and Alice's input as $E(s_1), \dots, E(s_n)$.

Having Bob choose an encryption function and using the previous protocol over the encrypted domain has two advantages. The main advantage is that Alice is not able to control the groups that a possible p maps to, and if E is a strong encryption function it is unlikely that a small group will exist. A secondary advantage is that in order for Alice to compute these groups she must encrypt every value in U , which is tractable for a few Bobs but not for all such Bob entities. This scheme requires $O(1)$ communication setup cost, and requires $O(\log n)$ comparison rounds and $O(\log n)$ comparisons. Thus this scheme requires $O(\log n \log \log N)$ rounds and $O(\log n \log N)$ communication. No communication is necessary to perform updates.

4.7 Protocol 7: Eliminating the Group Problem

The previous protocol still allows Alice to place p into a smaller group than the entire population. If Alice knows some out-of-band information then it could lead to undesirable consequences. For example, suppose Alice knows that two transactions have the same identity. Then she can take the intersection of the sets of calculated users to reduce the set to a very small group. This problem is removed in this section. The key problem is that Alice knows the search path in the tree and knows how the tree is assembled. If we offload part of this information to Bob then Alice cannot compute such information. We do this by adding a one-time setup cost (for each Bob) of $O(n \log N)$ to the system. The protocol is as follows:

1. **Setup Phase** Alice chooses a commutative cryptographic encryption scheme with encryption key E_A . Alice sends Bob $E_A(s_1), \dots, E_A(s_n)$. This need only be done once for each Bob.
2. **Search Phase** Bob chooses a commutative encryption scheme with encryption key E_B . Bob computes $E_B(E_A(s_1)), \dots, E_B(E_A(s_n))$ for himself and sends Alice $E_B(p)$
3. Alice computes $E_A(E_B(p)) = E_B(E_A(p))$ and then her and Bob engage in the protocol described in Section 4.5 (i.e., the binary search). For the search Bob learns the results of the *Secure Comparisons*, but the result of the *Secure Comparison* at the leaf node is split between Alice and Bob. The *Equality with Selection* protocol can be used to have Alice learn p if these encryptions match, or an invalid identity if it does not match.

The communication setup cost is $O(n \log N)$, and a search requires $O(\log n)$ comparison rounds and $O(\log n)$ comparisons. Thus this scheme requires $O(\log n \log \log N)$ rounds and $O(\log n \log N)$ communication. For Alice to insert a new suspect she must perform $O(\log N)$ communication (she must tell Bob which element to add), and to do a removal she must perform $O(\log n)$ communication (she must tell Bob which item to remove).

Theorem 4-7: The above protocol protects privacy.

Proof: We must consider the information learned by Alice and Bob. Bob learns a range for the value of $E_A(E_B(p))$ but he does not know E_A so this information is computationally private. Similarly Alice learns the value $E_A(E_B(p))$ but does not know E_B so this is computationally private. Alice learns nothing from the *Secure Comparisons*, and she learns only the result of the equality check that Bob performs at the leaf node. **QED**

4.8 Protocol 8: Scalability through Semi-trusted parties

The previous protocol mitigates the privacy concerns, but requires that Alice keep state information for each Bob and that she engage in a protocol with $O(\log n)$ comparison rounds for each identity that needs to be checked. This is obviously not scalable; one way to increase scalability would be to make several copies of Alice, but this increases the probability of data corruption (inconsistencies). In this section we propose to add a set of semi-trusted third parties (let Ursula be such a party). Ursula (and other parties like her) is only trusted to the extent that she will not collude with Alice or Bob (as will be seen below, we do limit the power of

colluding parties). This protocol is not more efficient than previous protocols, but Alice is able to offload most of her work to Ursula. This increases scalability since there can be multiple Ursulas without having the negative side effects of multiple Alice's.

Setup Phase:

1. Alice chooses a commutative cryptographic encryption scheme with encryption key E_A . Alice sends Ursula $E_A(s_1), \dots, E_A(s_n)$. This need only be done once for each Ursula (it is wise to change the key every once in awhile though).
2. Ursula forwards this information to each Bob.

Search Phase:

1. Bob chooses a commutative encryption function E_B and computes $E_B(E_A(s_1)), \dots, E_B(E_A(s_n))$, and then sends Alice $E_B(p)$.
2. Alice computes $E_A(E_B(p))$, and then chooses a hash function H that maps this value to a range that is smaller than N . It should be such that the probability of many values in S being mapped to the same value is relatively small, but there should be many values in U that are mapped to the same value. She sends H to Bob and computes $H(E_A(E_B(p)))$ which she sends to Ursula.
3. Bob computes $H(E_B(E_A(s_1))), \dots, H(E_B(E_A(s_n)))$. Ursula and Bob engage in the protocol described in Section 4.5 (i.e., the binary search) where Bob's inputs are $H(E_B(E_A(s_1))), \dots, H(E_B(E_A(s_n)))$ and Ursula's is $H(E_A(E_B(p)))$. For the search Bob learns the results of the comparisons, and when the leaf node is reached Bob will know a set of values S_p that could be equal to p (this is computed by taking all values in the bucket that p maps to). Note that $S_p \subseteq \{E_B(E_A(s_1)), \dots, E_B(E_A(s_n))\}$.
4. Alice and Bob engage in the last step of protocol 4.7 with Bob's input as S_p and Alice's input as $E_A(E_B(p))$.

The above scheme requires the same communication as the previous scheme. However, Alice only needs to pay the setup cost once per Ursula. Furthermore, in most cases she only needs to do $O(1)$ encryptions and $O(1)$ secure equality check, which is much more scalable than the previous solution. However in the unlikely event that the hash function maps many values to the same bucket, then she will have a similar communication overhead as the previous protocol.

Privacy of scheme:

1. Alice learns nothing if $p \notin S$ assuming the encryption functions are secure. This is because she learns $E_B(E_A(p))$ which is protected by encryption. Furthermore, she learns nothing more in the last step then she does in the final step of Protocol 7 (see Theorem 4-7).
2. Bob learns nothing about membership of p . This is true since he learns nothing more than he did in the previous protocol.
3. Assuming the encryption functions are secure, Ursula learns nothing about p or its membership.

4. If Alice and Ursula join forces they cannot learn anything about p without Bob's help, since it is hidden with E_B which neither know.
5. Bob and Ursula cannot determine if p is in Alice's set, and although there are cases where they could deduce that $p \notin S$ they cannot do this without Alice's help, which prevents an off-line attack that exhaustively tries all of U .

4.9 Summary of Schemes

The protocols in this section are listed in an evolving order, i.e., each protocol is developed by fixing a problem listed in the previous protocol. Tables summarizing the changes are listed in the appendix. We present a quick summary of these schemes in this Section. The initial privacy preserving protocol was not efficient in that it required the entire list of items be transmitted for each search. Our schemes remove the need to do this level of communication for every search, but require a similar amount of communication as a one-time setup cost. The scheme in Section 4.3 did this, but it allowed profiling of users (i.e., Alice could determine if the same customer visited the same Bob twice) and had scalability issues since Alice must keep a copy of her suspect list for each Bob. To alleviate the communication cost a binary search was introduced in Section 4.5, which introduced a new privacy problem in that Alice could place the identity of a search into a group that was smaller than the full population. In Section 4.7 this grouping issue was fixed, but the protocol was not scalable since it required Alice to do engage in several communication rounds with Bob for each identity. Thus in Section 4.8, a protocol that allowed Alice to offload most of her work to an untrusted third party (Ursula) without sacrificing any privacy if Ursula and Bob do not collude. Furthermore, even if Ursula and Bob were to collude, their power would be limited.

5. MONITORING ALICE

What if Alice started behaving as if S contains identities for which she has no authorization to monitor? Or, as Juvenal said, "Sed quis custodiet ipsos custodes?" ("But who watches the watchers?") [15]. If Alice were to do this, then she would be breaking the law, as opposed to the situation without privacy-preserving surveillance technologies (in which Alice would have all the intrusive surveillance data as a normal part of carrying out her duties). Privacy-preserving surveillance makes invalid the excuse that "Alice cannot do her job without intrusive surveillance", and makes illegal that which would otherwise be done routinely. Having said this, although it may be cumbersome to do so, there are cryptographic techniques that can be used to prevent Alice from the above-mentioned "adding extra entries to S " attack. In this section we propose a scheme for mitigating this problem, assuming the protocol in section 4.8 is used to perform the monitoring.

To monitor Alice's set S we propose introducing a third party Monty. The goal of this will be that Alice should not be able to modify S without Monty's knowledge. Of course, this would not work if Alice and Monty colluded and it does add another party that will know the actual value of S . However, it adds a level of protection to prevent Alice from unauthorized monitoring. We assume there is some mechanism for controlling who should go into Alice's list;

this probably will take the form of a warrant. In order for Monty to verify that $y \in S$ is correct, this warrant (denote the warrant for y as $W(y)$) should contain information that proves that y is a valid suspect and some expiration information about the warrant. We insert the following protocol during Alice’s setup phase with Ursula (for the protocol in Section 4.8):

1. Alice sends Monty $(y, W(y), E_A(y))$ for each $y \in S$, where E_A is Alice’s commutative encryption key for the protocol in Section 4.8.
2. Monty verifies the warrants and then signs (using a signature scheme S_M) a certificate of the form: $S_M(E_A(s_1), \dots, E_A(s_n), T)$ where T is the timeout for this information, and sends this back to Alice.
3. Alice passes this information to Ursula who verifies it before sending it to Bob.

Remarks: The above protocol makes Alice accountable to Monty whenever she creates S , and as long as these parties do not collude she will not be able to add arbitrary suspects to S . Now to update S Alice must go through the process again to get a different certificate, and thus Monty will be able to check Alice’s updates of S .

6. PROTECTING BOB’S INTERESTS

The protocols above protect the identities of people not in Alice’s suspect list S , and they prevent Bob from obtaining Alice’s set S . However, it does not prevent Alice from gathering information about Bob’s site. Although the identities are obscured in the transactions Alice still learns information of the following form: “ m people bought fertilizer from Bob on Friday”. This violates Bob’s right to privacy; we assume that Bob would not mind if Alice knew that a certain suspected criminal used his services. However, we cannot assume that Bob desires Alice to know statistical information about his site. In this section we propose a fix to this problem (we assume the protocols in Sections 4.7 or 4.8 are used). Bob chooses a private key encryption system, and sends Alice the transaction information including the identity encrypted with this key. At the final stage of the protocols, Alice will either learn the private key that Bob used (if the identity is a suspect) or a predefined value that will not decrypt the message. Now the details of Bob’s transactions are hidden, but the volume of transactions are still known. However, this is more acceptable than revealing individual transaction details.

7. CONCLUSIONS

In this paper we introduced protocols for performing privacy preserving electronic surveillance. This system could be used to monitor suspected terrorists, deadbeats, or criminals. We believe this paper is a first step towards a comprehensive set of privacy-preserving surveillance technologies. It would definitely be better if we lived in a world where surveillance is unnecessary, but the reality we live in makes surveillance a necessity, and this surveillance might then as well be of the privacy-preserving kind rather than of the intrusive kind. Furthermore, this system (and any such system) cannot guard against people that use alternate identities, but it can be used to “raise the bar” for

avoiding surveillance. Here we describe how this paper can be extended in the future:

- An implementation of this scheme outlined in this paper to determine if the schemes are truly capable of handling a large number of searches.
- Modify the protocols to include non-exact matchings (as in [8]) for a large database, which would have various applications like a DNA dragnet.
- In this paper we assumed that Alice knows a list of suspects. Another direction of research would be to develop privacy preserving suspect determination, i.e., Alice would engage in protocols with several databases to look for “suspicious behavior”.

8. REFERENCES

- [1] J. Biskup and U. Flegel, “Transaction-Based Pseudonyms in Audit Data for Privacy Respecting Intrusion Detection,” *Lecture Notes in Computer Science, Vol. 1907*, Springer Verlag, Proc. 3rd International Workshop on the Recent Advances in Intrusion Detection (RAID), Toulouse, France, October 2000, pp. 28–48.
- [2] C. Cachin, S. Micali and M. Stadler. Computationally private information retrieval with polylogarithmic communication. *Advances in Cryptology: EUROCRYPT '99, Lecture Notes in Computer Science*, 1592:402–414, 1999.
- [3] 2000 census data: www.census.gov
- [4] B. Chor and N. Gilboa. Computationally private information retrieval (extended abstract). In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, El Paso, TX USA, May 4-6 1997.
- [5] B. Chor, O. Goldreich, E. Kushilevitz and M. Sudan. Private information retrieval. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, Milwaukee, WI USA, October 23-25 1995.
- [6] C. Clifton, personal communication
- [7] G. Di-Crescenzo, Y. Ishai and R. Ostrovsky. Universal service-providers for database private information retrieval. In *Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing*, September 21 1998.
- [8] W. Du and M. J. Atallah. Protocols for secure remote database access with approximate matching. In *7th ACM Conference on Computer and Communications Security (ACMCCS 2000), The First Workshop on Security and Privacy in E-Commerce*, Athens, Greece, November 1-4 2000.
- [9] Y. Gertner, S. Goldwasser and T. Malkin. A random server model for private information retrieval. In *2nd International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM '98)*, 1998.
- [10] Y. Gertner, Y. Ishai, E. Kushilevitz and T. Malkin. Protecting data privacy in private information retrieval schemes. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, Dallas, TX USA, May 24-26 1998.

- [11] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 218–229. ACM Press, 1987.
- [12] O. Goldreich. Secure multi-party computation. Working Draft, 2000.
- [13] S. Goldwasser. Multi party computations: past and present. In *Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing*, pages 1–6. ACM Press, 1997.
- [14] Y. Ishai and E. Kushilevitz. Improved upper bounds on information-theoretic private information retrieval (extended abstract). In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, Atlanta, GA USA, May 1-4 1999.
- [15] Juvenal, Satires, VI, 347
- [16] K. Kurosawa and W. Ogata. Bit-slice auction circuit. In *ESORICS 2002*, LNCS 2502, pages 24–38, 2002.
- [17] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings of the 38th annual IEEE computer society conference on Foundation of Computer Science*, Miami Beach, Florida USA, October 20-22 1997.
- [18] B. Schneier. *Applied Cryptography – Protocols, algorithms, and source code in C*. John Wiley & Sons, Inc., 1996.
- [19] A.C Yao. Protocols for secure computation. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 160–164, 1982.

APPENDIX

The appendix on the next page contains tables summarizing the schemes presented in this paper.

A. PROTOCOL SUMMARY TABLES

This appendix contains tables summarizing the schemes in Section 4. Table 1 summarizes the communication complexity of various operations for each scheme. Table 2 briefly summarizes the protocols described above.

Table 1: Summary of communication complexities

Protocol	Setup	Search	Rounds	Insert	Delete
4.1	0	$O(\log N)$	1	0	0
4.2	0	$O(n \log N)$	$O(1)$	0	0
4.3	$O(n \log N)$	$O(\log N)$	$O(1)$	$O(\log N)$	0
4.4	0	$O(n \log N)$	$O(\log \log N)$	0	0
4.5	0	$O(\log n \log N)$	$O(\log n \log \log N)$	0	0
4.6	$O(1)$	$O(\log n \log N)$	$O(\log n \log \log N)$	0	0
4.7	$O(n \log N)$	$O(\log n \log N)$	$O(\log n \log \log N)$	$O(\log N)$	$O(\log n)$
4.8	$O(n \log N)$	$O(\log n \log N)$	$O(\log n \log \log N)$	$O(\log N)$	$O(\log n)$

Table 2: Summary of protocols

Protocol	Summary
4.1	No privacy with minimal communication
4.2	Full privacy with $O(n \log N)$ communication
4.3	Has $O(n \log N)$ setup cost, but $O(\log N)$ search cost. Furthermore, this scheme allows profiling.
4.4	Is another scheme with full privacy but uses secure comparisons so that profiling can be eliminated.
4.5	Introduces the usage of a binary search, but allows Alice to group users.
4.6	Partially fixes the grouping problem by eliminating the predictability of the groups with encryption.
4.7	Fixes the grouping problem by reversing Alice's and Bob's roles. However, this scheme requires Alice to do substantial work for each comparison.
4.8	Introduces an untrusted third party Ursula that allows Alice to offload her work, and thus increasing the scheme's scalability.