Private Combinatorial Group Testing

Mikhail J. Atallah[*]
Department of Computer Science
Purdue University
mja@cs.purdue.edu

Keith B. Frikken
Computer Science and Systems Analysis
Miami University
frikkekb@muohio.edu

Marina Blanton
Department of Computer Science and Engineering
University of Notre Dame
mblanton@cse.nd.edu

YounSun Cho[†]
Department of Computer Science
Purdue University
cho52@cs.purdue.edu

ABSTRACT

Combinatorial group testing, given a set C of individuals ("customers"), consists of applying group tests on subsets of C for the purpose of identifying which members of C are infected (or, more generally, defective in some way). The outcome of a group test reveals only the presence or absence of infection(s) in that group, but a number of group tests exactly identifies all infected members.

Although the main motivation for group testing is economic – it drastically cuts down the number of necessary tests – it has an interesting privacy side-effect, namely, that each individual customer is "hiding in a crowd" (the groups within which it is being tested). This privacy side-effect is currently thrown away because the analysis that pinpoints who is infected is carried out by the same entity that prepared the test samples. This paper gives a protocol in which these two duties are separated between Alice and Bob: The protocol informs each customer who is infected privately, and without either Alice or Bob learning who is infected. An interesting feature of our protocol is that a customer need not have any computational power, i.e., the customer can be notified by mailing her (possibly paper copies of) two random strings – one from Alice and one from Bob – so all she has to do is visually check whether these two strings are equal or not.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; E.3 [**Data**]: Data Encryption; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems.

General Terms

Security, Algorithms, Design.

Keywords

Privacy, Secure Protocol, Group Testing, Integrity Verification.

[*]Supported in part by Grants IIS-0325345 and CNS-0627488 from the National Science Foundation, and by sponsors of the Center for Education and Research in Information Assurance and Security.

[†]Supported by Grants IIS-0325345 and CNS-0627488 from the National Science Foundation.

1. INTRODUCTION

Combinatorial group testing on a set C is used to efficiently test which members of the set satisfy a specific test. This is done by combining the members into groups and running such tests on the groups rather than on individual entries. Such group tests, whether of blood samples or of digital objects, then reveal whether the elements of a group are *all* "clean" – not infected in the case of blood, not corrupted in the case of digital objects. From the outcomes of remarkably few such group tests, it is possible to infer which of a large set of n items are clean and which are defective.

The main application of group testing is to situations where (i) it is expensive to individually test every one of a large number n of samples (or, in the digital world, to individually monitor the integrity of every one of a large number n of data items, or to individually watch for anomalies in many event streams); and (ii) it is necessary to pinpoint precisely which are defective among the n samples (or data items, event streams, etc), based on a relatively small number of group tests. This was the original motivation for combinatorial group testing, as formulated for testing blood supplies during World War II for the syphilis antigen [3]. The cost of identifying which of n blood samples is tainted, can be dramatically decreased by applying tests to judiciously chosen subsets of the samples – given any constant upper bound on the number of tainted samples, a logarithmic (in n) number of tests suffice. A test consists of taking a few drops from a subset of the samples, mixing them, and applying a test to the mix, where a test outcome is "bad" if and only if the corresponding subset contains one or more infected blood samples. A more recent motivation for CGT has been to integrity verification [9], with the following analogies:

- In data integrity monitoring, the cryptographic hash of a subset of n digital items is analogous to a mix of blood drops from a subset of n blood samples, and comparing such a hash to what it is supposed to be, is analogous to applying a blood test to the mix of blood drops (a mismatch between the computed hash and the stored hash indicates that there is at least one corrupted element in the subset for that hash).

- In anomaly detection, an event stream may consist of events from n distinct sources, and instances of the detection mechanism are used for monitoring substreams that correspond to different subsets of these n sources (each instance learns what is normal for its monitored substream, and it does the monitoring without having to store all of its substream because there is not enough space for that). A substream tagged as "anomalous" implies that at least one of its constituent sources is behaving anomalously.

There are many other applications of combinatorial group testing in the digital world (see Section 5 for more of these), but for the sake of definiteness the rest of this paper uses the language of blood tests for n customers. We do so because privacy-preserving blood testing is of inherent interest in its own right, but it should be understood that our results have relevance beyond privacy-preserving blood testing (we further clarify this relevance later in this section).

It is not hard to see that there are situations where the leakage of an outcome of "infected" for a customer is a source of possible embarrassment, humiliation, or even more tangible consequences (possibly becoming uninsurable or less employable). A similar statement holds for entities that are not individuals – a "corrupted" outcome for their digital objects or event sequences could be a source of embarrassment and loss of reputation/goodwill, possibly triggering lawsuits or a drop in stock market value. This is the basic motivation for our work, which aims at providing a way to carry out group tests but without anyone other than the customer learning of their own diagnosis.

This kind of protocol where only the customer is aware of her test outcome is advisable as a risk-mitigation technique even in cases where most customers would feel comfortable trusting a single facility (Alice) with all four steps of blood sample handling: Collection, mixing, testing, inference of infecteds. That is, not only could such a need for risk mitigation come from the small percentage of Alice's customers who are demanding when it comes to privacy, but Alice herself may insist on using it, e.g., because it decreases her liability insurance premium (alternatively, the insistence that Alice not be completely trusted with such critical data may come from Alice's insurance company). Because no system or network is perfect, it is wise to recognize that the privacy of Alice's data could be breached (through a break-in, spy-ware, insider misbehavior, etc.) and to use technologies (such as our protocol) that make the consequences of such a compromise less disastrous.

Therefore, in our solution this task is divided between two parties – a data collection center Alice and a testing facility Bob – neither of which is entrusted with the result of the computation (which is to determine who is infected). In privacy-preserving blood testing this separation of duties is not far from current practice, as it corresponds to what often happens today in clinical environments: The Alice facility, where customers' blood samples are taken (possibly in the context of a blood donation drive), is not physically equipped to carry out the sophisticated and expensive testing, which is done at a remote facility Bob that is equipped for testing (that is, Alice and Bob already exist, and our proposal is merely for a change in the way they interact). The relevance of our techniques for cybersecurity, however, needs some further elaboration. As already stated, in integrity verification the equivalent to Bob's testing of a "blood mix" would be Bob's verification that the Alice-computed cryptographic hash of the concatenation of a group of records or files (the "blood mix") matches the signed version of that hash that is pre-stored securely with Bob. The storing of the signed "expected" values of the hashes with Bob would occur at system set up time, well before any run of our protocol – the signature would be carried out by a trusted authority (not by Bob) and Bob (but not Alice) would be provided with the signed hashes. In anomaly detection the event sub-streams would be pseudonymized before being mixed in groups and sent for anomaly analysis to Bob; events from the same source have different pseudonyms in the various groups of which they are part. Bob uses its own proprietary (and possibly computationally expensive) technology to monitor each group for the presence of anomalies. One of the drawbacks of anomaly detection technologies becomes a privacy advantage in this case: Unlike signature-based intrusion detection, which is capable of providing precise reasons why it sounded an alarm, an anomaly-based system typically does not provide such precise reasoning on why its conclusion of "anomaly" was reached.

Our contribution

We give a protocol for combinatorial group testing, that is privacy-preserving in the sense that it informs the customers of whether they are infected, without any other entity learning this. A distinctive feature of our protocol is that *the customers do not need to have any computational power at all* and are not required to have access to a computer: A customer obtains two random strings and learns whether it is infected or not by visually comparing these two strings for equality.

While ensuring the privacy of customers' outcome, we also minimize the computation needed by Alice and Bob – our solution is computationally efficient for both Alice and Bob even when the number of customers is large.

Additionally, we provide enhancements to the protocol that allow the detection of cases when the number of infected customers exceeds the assumed upper bound for it, and that lower the number of operations performed in the protocol.

We prove the security of our solution under the assumption that the underlying primitives used (such as encryption and random permutations) are secure. We assume that players Alice and Bob will follow the protocol and will not collude with each other, but they may collude with customers.

The rest of this paper is organized as follows. Section 2 presents the framework considered. Section 3 presents the protocol, its analysis, and extensions. Section 4 gives results from our experimental implementation of the protocol. Section 5 describes related work, and Section 6 concludes.

2. FRAMEWORK

This section sets forth the framework and security model we use. It also defines notation that is used throughout the paper.

2.1 Group testing background

Here we review the terminology and some known results from combinatorial group testing, that are needed in the rest of the paper. As mentioned earlier, combinatorial group testing (CGT) aims to perform group tests on subsets of a given set C to identify infected elements in C. The outcome of a group test is "contains at least one infected item" or "contains no infected items." We are interested here in *non-adaptive* CGT, in which all the subsets to be tested have to be decided ahead of time, i.e., before any subset is tested. In adaptive CGT, by contrast, the next subset to be tested can be chosen based on the outcomes of the previous tests. Non-adaptive is more appropriate for our framework – for example, in integrity verification, the decision of which subsets of n records will have their hashes computed and stored, has to be made ahead of time and before any compromise in integrity has occurred (hence before learning the outcomes of any tests).

There are known constructions that, given an upper bound d on the number of possible infected elements in C, can pinpoint the (at most d) infected items using a remarkably small number m of tests (that is, m subsets of C are tested). For example, when d is constant, $m = O(\log n)$ tests suffice. More specifically, the best known general-purpose adaptive schemes use $m = O(d \log(n/d))$ tests, whereas the number of tests used by the best known general-purpose non-adaptive schemes is $m = O(d^2 \log n)$ [4].

We now review the specific construction that we use here. An $m \times n$ Boolean matrix M is *d-disjunct* [4] if, for any $d+1$ columns one of which is *designated*, there always exists a row with a 1 in the designated column and 0's in the other d columns. Given a d-disjunct Boolean matrix M, a non-adaptive combinatorial group testing scheme consists of simply performing the test indicated by each row i of M (that is, test the subset S_i corresponding to the columns containing 1 in that row i). The results of these m tests are then used as follows:

1. Initialize all n items as being infected.

2. For every row i whose test's outcome was clean (i.e., no infected is in the subset S_i), mark all the elements of S_i as being clean.

3. The items not marked clean by the time this process ends are infected.

The correctness of this algorithm follows from M being d-disjunct: Any clean item x cannot fail to be marked as clean, because no matter which the d infected items are, there exists an S_i that contains x and none of the d infected ones, and this clean S_i will in turn cause x to also be marked clean.

It is well known [4] that a d-disjunct matrix M can be constructed in $O(mn)$ time by setting each of its entries to 1 with probability $1/(d+1)$. See [9] for how this can be done using only $O(d^3 \log n \log d)$ random bits, with the resulting m being $O(d^2 \log n)$.

2.2 Cryptographic background

We now briefly review a cryptographic primitive used in the protocol. Our protocol utilizes a public-key semantically secure additively homomorphic encryption scheme such as Paillier [13]. Recall that such encryption makes it possible to carry out certain computations on encrypted data. In particular, given messages m_1 and m_2 and encryption key k, we have: $E_k(m_1) \cdot E_k(m_2) = E_k(m_1 + m_2)$, and therefore $E_k(m_1)^{m_2} = E_k(m_1 \cdot m_2)$.

2.3 Security model

As was mentioned above, we rely on two entities who are not expected to collude to perform the testing: A data collection center Alice and a laboratory facility Bob. For each customer, Alice receives the customer identifying information and the blood sample, but does not have equipment to run the tests. Bob, on the other hand, does not have access to customers' blood samples, but rather runs tests on mixes comprised of samples from many customers. Furthermore, we assume Bob can neither identify the customers from a specific blood mix, nor learn the number of customers in the blood mix (i.e., all mixes contain approximately the same amount of blood). This last assumption is certainly true in integrity verification because a "blood mix" in that domain is a cryptographic hash, which consists of a fixed number of bits (independent of the number of customers in the mix). But it is also true in the physical blood situation, because the expected number of customers in a blood mix is the expected number of 1s in a row of the d-disjunct matrix M, which is $n/(d+1)$ because each entry of M has probability $1/(d+1)$ of containing a 1.

2.3.1 *Security objective*

We require that, under the adversarial model described below, it is infeasible for any party to compute the result of the test for a customer c_i. More formally, we require that no protocol participant can gain any information during the protocol execution, other than its own inputs and the outputs it is supposed to learn from the protocol (e.g., its own infection status). In other words, a participant's view of the protocol execution can be simulated given the inputs and outputs alone.

In addition to showing that the protocol execution does not leak information to the participants, we also need to guarantee that recovering a customer's status from the inputs and outputs of the protocol is difficult. In other words, it should be difficult for Alice or Bob to recover the status of a customer c_i after the execution of the protocol, even in cases where all the other customers voluntarily reveal their own infection status to Alice and Bob.

2.3.2 *Model of adversary*

If we let $C = \{c_1, \dots, c_n\}$ denote the set of customers, our model of the adversary is then as follows.

- Alice and Bob do not collude against a customer.
- Alice may collude against a customer c_i with (possibly all) the other customers.
- Bob may collude against a customer c_i with (possibly all) the other customers.

Our protocol will ensure that, assuming that Alice and Bob do not collude with each other, it is infeasible for any party to compute the result of the test for a customer c_i. As will become clear from the protocol description, the customers are not an active part of the protocol, and colluding with them during the protocol does not provide any advantage against the intended victim c_i.

3. PROTOCOL FOR NON-ADAPTIVE COMBINATORIAL GROUP TESTING

This section presents the protocol, its analysis, and implementation enhancements. As we desire to lower computational requirements of all entities involved, we pay a special attention to public-key cryptographic operations. Since the number of operations involved in CGT is unavoidably $O(mn)$, our private protocol will have to maintain this bound. The number of expensive modular exponentiation operations (which are also used to encrypt and decrypt data in homomorphic encryption schemes) can, however, be minimized. A more straightforward realization of this computation could require $O(mn)$ such operations, but our construction uses only $O(n)$ modular exponentiations and $O(mn)$ modular multiplications. Additionally, the $O(mn)$ bound on the number of multiplications can be lowered, as described in Section 3.3.2.

At a high level, the protocol works as follows: Upon receipt of m blood mixes from Alice, Bob tests each of these and sends to Alice an encrypted bit b_i for the ith mix, where $b_i = 0$ if the mix is infected and $b_i = 1$ otherwise. Using homomorphic properties of the encryption scheme, Alice computes for each customer $R\sum b_j + r$, where R and r are random values and the sum is over all mixes where the customer's blood was used. Now notice that a customer who is infected will have all of his bits b_i set to 0, resulting in $R\sum b_i + r = r$. Alice sends to Bob an encryption of r (with her key) and an encryption (with Bob's key) of $R\sum b_i + r$. Bob chooses two random values A and B for each customer. He decrypts the second value and forwards $A\left(R\sum b_i + r\right) + B$ to the customer. He also sends an encryption (with Alice's key) of $Ar + B$ to Alice. Alice then sends $Ar + B$ to the customer. Now the customer compares the strings received and concludes that she is infected if they match.

The detailed protocol steps are given below (where all arithmetic is modular and the modulus is either that for Alice's or Bob's homomorphic encryption, with the context making it implicitly clear which one it is).

3.1 Protocol specification

This section gives the main steps of the protocol – we postpone various implementation enhancements and algorithmic optimizations until section 3.3 (including them at this early stage would break the flow of the exposition).

Participants: A set of customers $C = \{c_1, \ldots, c_n\}$, a data collection entity Alice who can obtain blood samples from customers and prepare mixes, and a mix-testing entity Bob who can tell whether a mix is infected or not.

Input: An upper bound d on how many members of C can be infected. An $m \times n$ d-disjunct Boolean matrix M known to Alice but not to Bob. Linked lists $S_1, \ldots, S_m$ where S_i contains the column positions of row i of M that are 1. Linked lists $V_1, \ldots, V_n$ where V_j contains the row positions of column j of M that are 1.

Output: Each customer $c_i \in C$ learns whether he is infected. Neither Alice nor Bob learn which customers are infected.

Protocol steps:

1. Alice generates a private-public key pair for a homomorphic semantically secure encryption scheme [13]; we denote such encryption using Alice's public key as E_A. Similarly, Bob generates his own pair, and we denote encryption with Bob's public key by E_B.

2. Alice randomly assigns each customer c_i to a column of M; for the sake of notational simplicity, we re-name the customers so that c_i is assigned to column i.

3. Alice collects the n blood samples. For each of S_1, …, S_m in turn, Alice prepares a mix μ_i of the blood samples according to S_i: If column j is in S_i then the blood sample of the customer assigned to column j is a part of μ_i. Alice sends these m mixes (in the order μ_1 to μ_m) to Bob who tests each of them and obtains a bit b_i from testing μ_i: If μ_i is infected then Bob sets $b_i = 0$, otherwise Bob sets $b_i = 1$.

4. For $i = 1, \ldots, m$ in turn, Bob encrypts b_i with his public key, obtaining $E_B(b_1), \ldots, E_B(b_m)$ which he sends to Alice.

5. For each $j = 1, \ldots, n$ in turn, Alice chooses two random numbers R_j and r_j from $\mathbb{Z}_{n_B}$, where n_B is the modulus used for E_B, and computes the following two quantities: $Y_j = r_j$ and
$$\begin{aligned} Z_j &= \left(\prod\nolimits_{i\in V_j} E_B(b_i)\right)^{R_j} \cdot E_B(r_j) = \\ &= E_B\left(\left(R_j \sum\nolimits_{i\in V_j} b_i\right) + r_j\right) \end{aligned}$$
Alice sends Z_j and $E_A(Y_j)$ to Bob.

6. For $j = 1, \ldots, n$ in turn, after Bob receives the Z_j and $E_A(Y_j)$ pair he decrypts Z_j to obtain
$$W_j = \left(R_j \sum\nolimits_{i\in V_j} b_i\right) + r_j$$
He then chooses two random values A_j and B_j from $\mathbb{Z}_{n_A}$, where n_A is the modulus used for E_A, and sends $A_j \cdot W_j + B_j$ (modulo n_A) to customer c_j. He also sends $\left(E_A(Y_j)^{A_j}\right) E_A(B_j) = E_A(A_j \cdot Y_j + B_j)$ to Alice.

7. For $j = 1$, …, n in turn, Alice decrypts the $E_A(A_j \cdot Y_j + B_j)$ received from Bob to obtain $A_j \cdot Y_j + B_j$ which she sends to customer c_j.

8. Each customer c_j compares the $A_j \cdot Y_j + B_j$ received from Alice to the $A_j \cdot W_j + B_j$ received from Bob in Step 7: If they are equal, then the customer c_j learns that she is infected, otherwise the customer learns nothing other than the fact that she is not infected.

In the above, Alice does $O(n)$ modular exponentiations and $O(nm)$ total work. Bob, on the other hand, performs $O(m+n)$ work. The communication costs of the protocol are also $O(m+n)$.

3.2 Security analysis

We will now show that Alice and Bob cannot infer significant information about whether an individual customer is infected. If Alice or Bob do not collude with other customers, then this is relatively straightforward (actually a simpler version of the above protocol would work). However, suppose Alice (resp., Bob) colludes with a set of participants C' and thus the adversary obtains all of Alice's (resp., Bob's) information along with the information on both sheets of paper for all customers in C'.

3.2.1 Alice's view

We first show that, given Alice's inputs and the infection status of all members in C', a simulated transcript can be generated that is computationally indistinguishable from Alice's view of the protocol. For each customer c_i Alice knows r_i, R_i, and V_i. She also knows several values encrypted with E_B (but these are easily simulated since E_B is a semantically-secure encryption scheme). However, she also learns the two following pieces of information: (i) $A_i r_i + B_i$ for all customers and (ii) $A_i\left(R_i \sum_{j\in V_i} b_j + r_i\right) + B_i$ for all customers in C'. A simulation algorithm can generate pairs of values (given infection status of members of C') as follows:

1. For infected customers in C', choose a random value x from $\mathbb{Z}_{n_A}$ (where n_A is the modulus of Alice's homomorphic encryption scheme) and output x for both pieces of information.

2. For non-infected customers in C', choose a random values x and y from $\mathbb{Z}_{n_A}$ and output x for the first piece of information and y for the other.

3. For every customer not in C choose a random value x from $\mathbb{Z}_{n_A}$ and output x for the first piece of information.

It is trivial to show that parts 1 and 3 are indistinguishable from their counterparts. For the second part, it is enough to show that for any value x and y ($x \neq y$) there is a single value of A_i and B_i where $A_i r_i + B_i = x$ and $A_i(R_i \sum_{j\in V_i}(b_j) + r_i) + B_i = y$. This is satisfied when

$$A_i = \frac{y-x}{R_i \sum_{j\in V_i} b_j} \quad \text{and} \quad B_i = x - \frac{r_i(y-x)}{R_i \sum_{j\in V_i} b_j}.$$

Thus such a value exists whenever $R_i \sum_{j\in V_i} b_j$ has an inverse (which will be true with all but negligible probability).

The above shows that Alice learns only information that can be deduced from the infection status of customers in C' and her input. However, given this information, we would like to determine whether it would be possible for Alice to infer information about honest customers. This is a valid concern because Alice knows the mapping between the customers and the sets V_j. Thus, if it is possible for Alice to infer the status of which samples are infected, she can use these mappings V_j to infer information as to which honest customers are infected.

Definitely, in our framework some inferences can be made. For example, if C' contains d infected customers, then Alice can infer that everyone else is not infected.[1] However, if C' contains at most $d-1$ infected customers, then she cannot determine if an individual customer c_s is infected as illustrated below.

- If c_s is infected, then Alice's view of the customers in C' is still consistent. Clearly, this could not change her view of the samples that contain an infected member of C'. Furthermore, since every non-infected member of C' is in at least one sample that does not contain c_s or any other infected members of C (otherwise the matrix would not be d-disjunct), her view of the non-infected members would not change.

- Similarly, if c_s was not infected, then clearly this would not affect her view of the non-infected members of C', but it would also not affect her view of infected members, because c_s is in at least one sample that does not contain any infected members of C' (otherwise the matrix would not be d-disjunct).

3.2.2 Bob's view

The proof that Bob's view is simulateable given his inputs is similar to Alice's proof. We first show that, given Bob's inputs and the infection status of all members in C' with whom he conspires, a simulated transcript can be generated that is computationally indistinguishable from his view of the protocol. For each customer c_i Bob knows A_i and B_i, and he knows b_i for $i \in [1, m]$. He also knows several values encrypted with E_A (but these are easily simulated since E_A is a semantically-secure encryption scheme). However, he also learns the two following pieces of information: (i) $R_i \sum_{j\in V_i} b_j + r_i$ for all customers and (ii) r_i for all customers in C'. A simulation algorithm can generate these pairs of values (given infection status of members of C') as follows:

1. For infected customers in C', choose a random value x from $\mathbb{Z}_{n_A}$ and output x for both pieces of information.

2. For non-infected customers in C', choose a random values x and y from $\mathbb{Z}_{n_A}$ and output x for the first piece of information and y for the other.

3. For every customer not in C' choose a random value x from $\mathbb{Z}_{n_A}$ and output x for the first piece of information.

It is trivial to show that parts 1 and 3 are indistinguishable from their counterparts. For the second part, it is enough to show that for any value x and y ($x \neq y$) there is a single value of R_i and r_i where $r_i = x$ and $R_i \sum_{j\in V_i} b_j + r_i = y$. This is satisfied when

$$r_i = x \quad \text{and} \quad R_i = \frac{y - r_i}{\sum_{j\in V_i} b_j}.$$

Note that R_i exists whenever $\sum_{j\in V_i} b_j$ has a multiplicative inverse in $\mathbb{Z}_{n_A}$. Since $x \neq y$, we know that $\sum_{j\in V_i} b_j$ is in $[1, m]$, and for any practical value of m this will be a coprime with any secure RSA modulus.[2]

[1] We are assuming that Alice knows for sure that there are d or less infected customers. In Section 3.3.1 we introduce an extension that aborts the protocol when more than d customers are found to be infected. If this extension is used, then this inference can always be made.

[2] We are assuming a homomorphic scheme such as Paillier where the modulus is an RSA modulus.

The above shows that Bob can only infer information about an honest customer's infection status if he can make that inference based upon his inputs alone. Bob's only knowledge from the protocol is the b_i values (he knows what samples were infected), but he does not know the mapping between customers and samples. However, from this information he may be able to deduce some information about the number of infected customers (e.g., lower and upper bounds). In some extreme cases, this may leak information. For example, if Bob knows that all samples are clean, then he can deduce that all customers are not infected. As another example, suppose only a few samples are infected, and, given the number of infected samples, Bob deduces that exactly one customer is infected (at least with high probability). Now if Bob colludes with all but one customer all of which are not infected, he can deduce that this honest customer is infected. However, extreme cases that lead to Bob identifying an honest customer as infected are unlikely in practice. That is, at most, Bob learns the number of infected customers. Thus to be able to pinpoint a customer as infected, Bob needs to collude with almost all customers, which will be unlikely in many environments.

We summarize what we have proved in the following theorem.

THEOREM 1. *As long as Alice and Bob do not collude with each other against a customer c_i, it is infeasible for any participant to infer the infection status of c_i.*

3.3 Implementation enhancements

In this section we consider various enhancements of our protocol. In section 3.3.1 a modification is presented that detects when there are more than d infected customers. Section 3.3.2 decreases the cost of computing all of the Z_j's. Section 3.3.3 gives a more efficient protocol for the special case when $d = 1$.

3.3.1 Handling more than d defectives

In the previous protocol, if there are more than d infected customers, then several false positives may occur. Bob may be able to detect such an overflow since he knows the number of infected samples, but it would be better to have a detection mechanism that invalidates all of the results when the threshold d is surpassed. Thus, here we show how it can be achieved. The main idea in our solution is as follows: Alice will transmit to Bob the number of marked customers (without revealing which customers are infected). If this is above the threshold d, then he aborts the protocol. Note that this reveals slightly more information to Bob in that he now knows exactly the number of infected customers instead of an estimate of the number of infected customers (which he obtained from knowing the number of infected samples). As to not clutter the exposition, we describe only the changes to the protocol:

- In Step 5 of the protocol, Alice also generates a value $D_j = E_B\left(\sum_{i \in V_j} b_j \cdot R'_j\right)$ for some random value R'_j. She transmits these values to Bob in a randomly permuted order (to hide which value corresponds to each customer).

- In Step 6, Bob decrypts the D_j values from Alice and counts the number of zeroes. If the number is larger than d, then Bob aborts the protocol. Otherwise, the protocol continues as before.

3.3.2 Algorithmic optimization

This sub-section gives an improved algorithm for decreasing the number of scalar multiplications needed for the computation of all the Z_j's. Specifically, we bring the number of arithmetic operations from $O(mn/d)$ to $O(d^2 n)$.

The computation of the Z_j's could be done naively, by computing each Z_j separately from the others. The number of scalar multiplications done by such an approach would then be proportional to the number of nonzero entries in the d-disjunct matrix M, whose expected value is

$$mn/(d+1) = O((d^2 \log n)(n/(d+1))) = O(dn \log n).$$

Had our $m \times n$ matrix M been an arbitrary Boolean matrix, the problem of minimizing the total number of scalar multiplications done for computing all the Z_j's would have been NP-hard. This can be proved by a straightforward reduction from the `ENSEMBLE COMPUTATION` problem [6], and in fact the NP-hardness would hold even if every V_j consisted of no more than 3 elements. However, our M matrix is generated by a very specific randomized construction, and has $m = O(d^2 \log n)$ rows. We show below how the computation time of the Z_j's can be brought down from $O(mn/d)$ to $O(d^2 n)$.

Recall that $m = \alpha d^2 \log n$ where α is a constant. Also recall that V_j contains the row positions of column j that are 1 and we will assume that each V_j is stored in αd^2 memory cells of size $\log n$ bits each. Let X_i denote $E_B(b_i)$ for $i = 1, \ldots, m$. The algorithm steps for computing $\prod_{i \in V_j} E_B(b_i)$ for each customer c_j, all of which are carried out by Alice, are as follows.

Input: $X_1, \ldots, X_m$ and $V_1, \ldots, V_n$.

Output: $Z'_1, \ldots, Z'_n$ where $Z'_j = \prod_{i \in V_j} X_i$.

Algorithm steps:

1. Partition the interval of integers $[1, m]$ into αd^2 chunks of size $\log n$ each, let I_k denote the kth such chunk. Our strategy will be to compute each Z'_j as the sum of αd^2 values $Z_{j,k}$, $1 \le k \le \alpha d^2$, where $Z_{j,k}$ is the contribution to Z'_j coming from the rows in row interval I_k.

2. For $k = 1$ to αd^2 in turn, compute each of $Z_{1,k}, \ldots, Z_{n,k}$ as follows.

   (a) For every one of the subsets of the rows in I_k, compute the product of the X_i's that correspond to the rows of that subset. The number of such subsets is $2^{|I_k|} = 2^{\log n} = n$, and the computation of the products for them can be done in $O(n)$ time by scheduling them in an order such that each product is obtained with one additional multiplication (i.e., multiplying one of the already-computed products by an X_i).

   (b) Store the n products computed in the previous step in an array of size n, so that the value of each such product can be read in constant time from the array.

   (c) Read from the array created in the previous step each of $Z_{1,k}, \ldots, Z_{n,k}$.

The time to compute all of $Z_{1,k}, \ldots, Z_{n,k}$ in the above algorithm is $O(n)$, and this must be done for αd^2 distinct values of k, resulting in an overall time complexity of $O(d^2 n)$ for the algorithm. For a constant d and/or large n, this compares favorably with the naive algorithm's $O(dn \log n)$ time complexity.

3.3.3 Better handling of the case $d = 1$

A particularly efficient $(\log n + 1)$-test deterministic CGT solution exists for the special case of $d = 1$ [3]. We first briefly describe it, and then point out how the protocol can be modified to implement it.

One of the tests performed is for a mixture of all the blood samples – it serves to determine whether there is contamination somewhere. The remaining $\log n$ tests are for determining which sample is corrupted, and are as follows. For $j = 1, 2, \ldots, \log n$, the jth test is for the mixture of those samples i for which the integer i has a 1 in the jth least significant bit of its binary representation. After obtaining the test results, we need to find out which sample is contaminated. To determine its number, the binary representation of such sample is constructed one bit at a time, as follows: For $j = 0, \ldots, \log n - 1$ in turn, if the jth test says "contaminated" then the jth bit of i is 1, and if it says "clean" then the bit is 0.

To illustrate this on an example, consider the case of 2000 samples $1, 2, \ldots, 2000$, of which only one is contaminated (assume it is the sample 676). In that case the test of the mixture of all the 2,000 samples says "contaminated," which indicates that there is a contamination. The other 11 ($= \log n$) auxiliary tests reveal which sample is contaminated, as follows. The 11-bit binary representation of 676 is 01010100100, and the sample 676 is therefore a part of the tests for bit positions 2, 5, 7, 9, and it is not a part of the tests for bit positions 0, 1, 3, 4, 6, 8, 10. The four tests that contain 676 will say "contaminated," whereas the other seven tests will say "clean." This implies that the contaminated sample i has (i) a 1 in bit positions 2, 5, 7, and 9, of the 11-bit binary representation of i; and (ii) a 0 in bit positions 0, 1, 3, 4, 6 ,8, 10.

To modify our protocol to take advantage of this faster algorithm, all that needs to be done is:

1. Change the V_j sets to reflect the way the tests are perform. In this case, $m = \log n + 1$ and for security reasons we do not have a special treatment for the first test.

2. In Step 6 of the protocol, change the formula for Z_j to

$$Z_j = \left(\prod_{i \in V_j} E_B(b_i) \cdot \prod_{i \notin V_j} E_B(1 - b_i)\right)^{R_j} \cdot E_B(r_j).$$

   Note that, as before, for an infected mix i b_i is set to 0 and b_i is set to 1 if the mix is clean. It is not difficult to verify that the above formula will produce the correct result for all customers in both cases when there are no infected customers and when there is a single infected customer.

4. EXPERIMENTAL PERFORMANCE EVALUATION

The purpose of this section is to demonstrate the performance and scalability of the protocol of Section 3. Our implementation was built and the experiments were run on a commodity hardware (a 2.67 GHz computer with 2GB of memory), which should be viewed as modest resources for data computing centers A and B. But even with such resources, our protocol scales well to a large number of customers. Also note that the protocol will not be invoked with high frequency, and a rather high overhead can be tolerated.

	n				
	2000	3000	4000	5000	6000
$d = 2$	548	577	598	614	627
$d = 3$	913	961	996	1023	1044

Table 1: The number of tests m necessary for varying values of n and d.

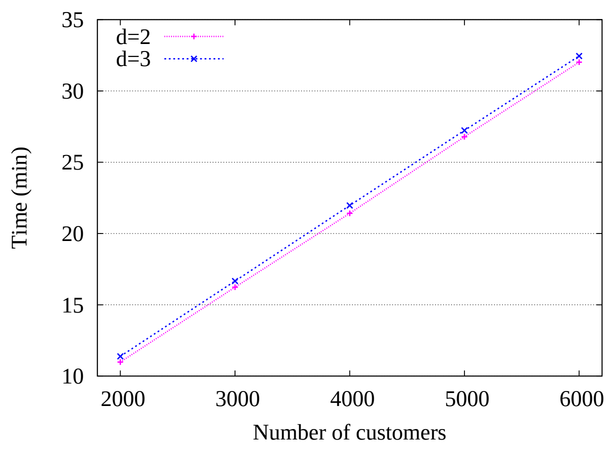
Our implementation was built using Paillier homomorphic encryption scheme [13] with 1024-bit modulus and the GMP large number library [1] and was written in C. Our Paillier implementation included an optimization for decryption operations as described in [13].

The results reported correspond to the computation carried out for the entire protocol (i.e., the computation performed by A and the computation performed by B). The communication is not included in the measurements and such overhead will depend on the type of communication link between the participants. We only note that Alice and Bob exchange $O(m+n)$ values during the protocol (and then distribute n values each to the customers at the end of or after the protocol).

In the measurements, we separate computation that can be performed in advance by Alice or Bob (pre-computation) from the computation that must be performed during the actual execution of the protocol. Table 1 shows the values of m used as a function of n and d. Figure 1 shows the amount of pre-computation needed for the protocol and the runtime of the protocol with different values of n and d (and the corresponding values of m). From the graphs we can see that both pre-computation and running times grow linearly with the number of customers. This growth is expected, but it is an interesting and somewhat unexpected result that increasing the threshold d had only a marginal impact on the performance (recall that increasing d even by 1 significantly increases m).

5. RELATED WORK

Several researchers have studied combinatorial group testing and its applications to cryptography and information encoding. The work of Colbourn et al. [2] and Du and Hwang [4] provide broad surveys. Stinson et al. [14] explored applications of group testing to key distribution in cryptography. The first work to use group testing for data integrity was due to Goodrich et al. [9], which established a connection between data forensics marking and a new reduced-randomness construction of a non-adaptive combinatorial group testing scheme. The present paper's analyses and experiments use the randomized construction of d-disjunct matrices given in [9]. This is the only overlap we have with [9], whose main focus was to use the randomized construction it introduced towards the forensic marking of data structures. Specifically, it used it for organizing the indexing structures of how data is stored so that alterations from an original version can be detected and the changed values specifically



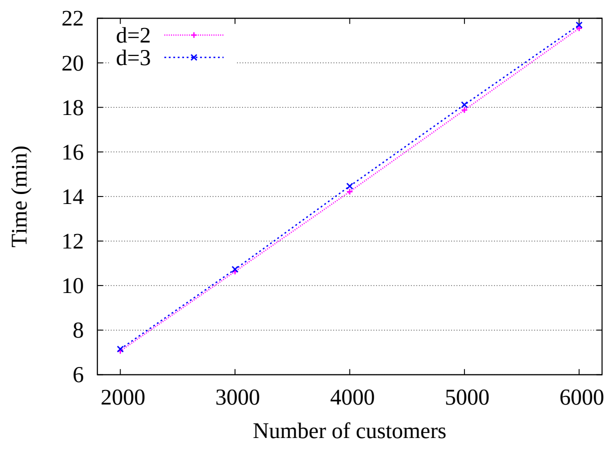


Figure 1: Pre-computation time (left) and runtime (right) of the protocol.

identified. It gave forensic constructions for several fundamental data structures, including binary search trees, skip lists, arrays, linked lists, and hash tables.

Much of the group testing literature is on adaptive group testing schemes, which generally make fewer total tests, in terms of d and n, than non-adaptive schemes. For example, the best known general-purpose adaptive schemes use $O(d \log(n/d))$ tests, whereas the number of tests used by the best known general-purpose non-adaptive schemes is $O(d^2 \log n)$ [4]. Adaptive schemes are clearly not applicable in the context of the present paper (as was the case in [9]). Another area where non-adaptive group testing schemes are more applicable is in DNA sequence analysis [11].

Another related area is that of Secure Multi-party Computation (SMC), which deals with computing a function over private inputs without revealing anything other than what can be computed from the result and some of the inputs alone. While there are general results stating that any function can be computed in such a manner [15, 8], these would lead to inefficient solutions for the PCGT problem. Our techniques for using the properties of homomorphic encryption to build a special purpose protocol is reminiscent of the techniques used in privacy-preserving protocols for: set operations [5, 10], scalar product [7], and stream searching [12].

6. CONCLUSIONS

We gave secure and private protocols for combinatorial group testing of n customers, where the test samples are prepared by Alice but the tests are carried out by Bob, and only the customer learns whether it is infected or not. In addition to the application areas mentioned in the above (blood testing, data integrity, event stream anomaly), we believe our techniques may be useful in providing privacy in the DNA analysis domain (see [11] for a detailed survey of the use of group testing in DNA analysis). This, however, requires further investigation because in DNA analysis group testing is used as one of many steps. We leave this exploration of DNA analysis privacy for future work.

7. REFERENCES

[1] The GNU multiple precision (GMP) arithmetic library. `http://gmplib.org`.

[2] C. J. Colbourn, J. H. Dinitz, and D. R. Stinson. Applications of combinatorial designs to communications, cryptography, and networking. In Walker, editor, *Surveys in Combinatorics*, volume 187 of *London Mathematical Society Lecture Note Series*, pages 37–100. Cambridge University Press, 1993.

[3] R. Dorfman. The detection of defective members of large populations. *Ann. Math. Statist.*, 14:436–440, 1943.

[4] D.-Z. Du and F. K. Hwang. *Combinatorial Group Testing and Its Applications*. World Scientific, 2nd edition, 2000.

[5] M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Proceedings of Advances in Cryptology - EUROCRYPT '04*, volume 3027 of *Lecture Notes in Computer Science*, pages 1–19, 2004.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, 1979.

[7] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikainen. On private scalar product computation for privacy-preserving data mining. In *The 7th Annual International Conference on Information Security and Cryptology (ICISC 2004)*, 2004.

[8] 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.

[9] M. T. Goodrich, M. J. Atallah, and R. Tamassia. Indexing information for data forensics. In *ACNS*, pages 206–221, 2005.

[10] L. Kissner and D. Song. Privacy-preserving set operations. In *Proceedings of Advances in Cryptology - CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, 2005. Full version appears at `http://www.cs.cmu.edu/~leak/`.

[11] H. Ngo and D.-Z. Du. A survey on combinatorial group testing algorithms with applications to dna library screening. In *Discrete Mathematical Problems with Medical Applications*. DIMACS Series, 55, American Mathematical Society, 2000.

[12] R. Ostrovsky and W. Skeith. Private searching on streaming data. In *CRYPTO*, volume 3621 of *Lecture*

Notes in Computer Science, pages 223–240, 2005.

[13] P. Paillier. Public key cryptosystem based on composite degree residue classes. In *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238, 1999.

[14] D. R. Stinson, T. van Trung, and R. Wei. Secure frameproof codes, key distribution patterns, group testing algorithms and related structures. *Journal of Statistical Planning and Inference*, 86:595–617, 2000.

[15] A. C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.