Department of Computer Science Technical Reports

Department of Computer Science

2009

Homomorphic Encryption based k-out-of-n Oblivious Transfer Protocols

Mummoorthy Murugesan

Wei Jiang

Erhan Nergiz

Serkan Uzunbaz

Report Number:
09-007

Homomorphic Encryption based *k*-out-of-*n* Oblivious Transfer Protocols

Mummoorthy Murugesan
Wei Jiang
Erhan Nergiz
Serkan Uzunbaz

CSD TR #09-007
September 2009

Homomorphic Encryption based k-out-of-n Oblivious Transfer Protocols

Mummoorthy Murugesan [1], Wei Jiang [2], Erhan Nergiz [1], and Serkan Uzunbaz [1]

[1] Purdue University, [2] Missouri University of Science and Technology

Abstract. Oblivious Transfer (OT) is an important cryptographic tool, which has found its usage in many crypto protocols, such as Secure Multiparty Computations [9], Certified E-mail [2] and Simultaneous Contract Signing [20]. In this paper, we propose three k-out-of-n OT (OT_k^n) protocols based on additive homomorphic encryption. Two of these protocols prohibit malicious behaviors from both sender and receiver. We also achieve efficient communication complexity bounded by $\mathrm{O}(l \cdot n)$ in bits, where l is the size of the encryption key. The computational complexity is same or better than the most efficient existing protocols. Due to the semantic security property, the sender cannot get receiver's selection. When the receiver tries to retrieve more than k values, the receiver is caught cheating with $1 - 1/m$ probability (Protocol II) or the receiver is unable to get any value at all (Protocol III). We introduce a novel technique based on the solvability of linear equations, which could find its way into other applications. We also provide an experimental analysis to compare the efficiency of the protocols.

Key words: Oblivious Transfer, Cryptographic Protocol

1 Introduction

In recent years, the tremendous amount of data collection has fueled concerns for information security and privacy. Many cryptographic schemes have been studied and proposed to accomplish tasks with minimal information disclosure. Protocols such as Secure Multiparty Computations [9], Certified E-mail [2] and Simultaneous Contract Signing [20] are few examples. Recent works on Digital Rights Management advocate privacy through oblivious transfer of digital contents to the sender [14]. In such protocols, oblivious transfer (OT) [19] is an important basic cryptographic component.

A more concrete example utilizing the OT protocol is as follows. Suppose Bob, a researcher, wants to download some valuable cryptography research papers which are sold by an authority, Alice. Bob wants to get k out of n research papers, which are very valuable for him. However, he does not want Alice to learn which research papers are retrieved in order to hide his topics of interest from Alice. In addition, Alice wants to ensure that Bob can get no more than the k number of papers he paid for. There comes the need for an OT protocol between Bob as the receiver and Alice as the sender.

There are two parties in an oblivious transfer protocol: a sender (e.g., Alice) who has a set of values and a receiver (e.g., Bob) who wants to get only certain values from Alice. Here, Alice wants Bob to obtain only a certain number of values, while Bob does not want to reveal to Alice any information about which values he chooses to get. The topic of OT has been extensively studied in many different forms using different cryptographic schemes. In this paper, we make use of homomorphic encryption that has semantic security property (e.g., Paillier [18]) to propose efficient k-out-of-n OT (OT_k^n) protocols.

There are many types of oblivious transfer protocols, each of which is based on different techniques and cryptographic schemes. The first oblivious transfer protocol was designed by Rabin [19], which was followed by other types of OT. In 1-out-of-n OT, the sender has n values and the receiver transfers one value. k-out-of-n OT is similar to 1-out-of-n OT where the receiver gets k of n values instead of one. In this paper, we focus on k-out-of-n OT. Although it can be achieved through k runs of 1-out-of-n OT, this increases time and communication complexity for most cases, and hence, it is costly.

We propose three efficient k-out-of-n OT protocols based on a probabilistic public key cryptosystem with additive homomorphic and semantic security properties. The semantic security property prevents the sender from getting the receiver's selections. The last two protocols (Protocol II and III) are designed in such a way that even a malicious receiver fails to get any advantage. Protocol II offers a high probability of detecting cheating receiver. Protocol III introduces a novel technique based on the feasibility of solving a system of k linear equations with more than k unknowns. Thus, any attempt to retrieve more than k values will prevent the receiver from getting any value at all.

The rest of the paper is organized as follows: Section 2 provides an overview on existing OT protocols and background information. In Section 3, we present three efficient OT_k^n protocols. Section 4 presents experimental results, and Section 5 concludes the paper with future research directions.

2 Background and Related Work

Rabin [19] first introduced OT_1^2, which is based on quadratic roots modulo a composite. After that, different versions of OT_1^2 were proposed in [6, 7]. Brassard et al. introduced OT_1^n named as all-or-nothing disclosure of secrets (ANDOS) inspired by OT_1^2 [3, 4]. This resulted in making the OT_1^n an open research problem in cryptographic protocol design.

The OT_k^n has become another open research problem following 1-out-of-n OT. Noting that these three problems can be reduced to each other, several works appeared based on this topic. In [21], Wu et al. stated that k-out-of-n t-bit string OT can be achieved applying Rabin's version of OT, $\mathrm{O}(kt \log n)$ times. In another version, OT_k^n is achieved using a OT_1^n scheme $\mathrm{O}(k)$ times [21]. Although there are efficient reduction protocols [16], there are numerous proposed OT_k^n schemes [5] that do not rely on existing OT_1^2 or OT_1^n protocols, giving better complexity than the reduction versions of these protocols.

The OT_k^n scheme suggested by Noar and Pinkas [16] has a computation complexity $\mathrm{O}(wk \log n)$ invocations of a basic OT_1^2, where w is a security parameter. The drawback of this scheme is that it works only when $k \leq n^{1/4}$. In [21], efficient homomorphic based OT_k^n was proposed for a condition when $k \geq n/\log n$. However, compared to our scheme, their protocols require interaction between sender and receiver, and a complex zero-knowledge paradigm is used to check that the receiver does not cheat to get more than k items. In a more recent work [5], efficient OT_k^n protocols were proposed based on the Diffie-Hellman Problem. The overall computation and communication complexity of this scheme are $\mathrm{O}(n)$ and $\mathrm{O}(nk)$, same as ours. However, their most secure protocol requires more than double the number exponentiations needed in our protocols.

The most efficient protocol based on homomorphic encryption is proposed in [13] by Malek and Miri. Let us denote this protocol as MM05-OT. This is an OT_1^n protocol that has $\mathrm{O}(\log n)$ communication complexity and $\mathrm{O}(n)$ computational complexity. This version of the protocol could be used to design an efficient OT_k^n with $\mathrm{O}(k \log n)$ communication and $\mathrm{O}(nk)$ computational complexity. However, a technical flaw in construction of this protocol (see Appendix 6.1) makes it produce wrong results almost all the time.

In all three protocols, we use a probabilistic public key encryption scheme (Paillier's cryptosystem [18]) with the additive homomorphic property. Such an encryption function has *semantic security* (defined in [12]), and is additive homomorphic. Appendix 6.2 gives a formal definition of such systems. We use $E(x)$ and $D(c)$ to denote the encryption and decryption functions respectively, where x is a plaintext, c is a ciphertext.

3 Protocols for k-out-of-n OT

The k-out-of-n OT is defined as follows: Suppose there are two parties, a sender and a receiver. The sender has a set of n values $v_1, \ldots, v_n$. The receiver retrieves k of these values without letting the sender know which values are retrieved. In other words, the transfer of k values is oblivious to the sender. Without privacy requirement, we can assume that the receiver sends a n-bit selection string to the sender. If $n = 5$, a selection string $X = \{0, 1, 0, 1, 0\}$ retrieves data values v_2 and v_4. The value 0 at positions 1, 3 and 5 means that the data values v_1,v_3 and v_5 are not selected. The sender can compute the component-wise product of X with data values, $Z_i = X_i * v_i$. Thus, $Z = \{0, v_2, 0, v_4, 0\}$ contains the data values retrieved by the receiver. We define a *well-formed selection string* with respect to n and k as follows:

Definition 1. *(Well-formed Selection String X_n^k) Let X_n^k be a n-bit selection string from the receiver. X_n^k is considered well-formed with respect to k-out-of-n protocol if the string X_n^k contains exactly k 1's and $n - k$ 0's.*

The simple protocol given above has two issues that need to be addressed to make the protocol oblivious. First, the receiver's selection string X should not be revealed to the sender. Second, the receiver should not be able to get more than k values from the sender i.e., X should be well-formed.

We can define two aspects of security in Oblivious Transfer protocols: 1) Sender's security and 2) Receiver's privacy. Sender's security is about making sure that the receiver gets only the k values, and no more values are released. Receiver's privacy is based on hiding the k selections from the sender so that the sender does not know which values are selected by the receiver. There are two behavior patterns that the sender and receiver may adopt.

1. **Semi-Honest** parties will follow the steps of the protocol; they may also process the outputs and the messages exchanged to gain extra information.
2. **Malicious** parties may try to break the protocol by not following the prescribed step, and deviating as necessary to gain advantage over the other party.

We now present three efficient k-out-of-n protocols. In Protocol I (Section 3.1), we consider a semi-honest receiver and a malicious sender. In Protocol II (Section 3.2) and Protocol III (Section 3.3), we provide security in the presence of malicious receiver and sender. Protocl III uses a novel approach based on the solvability of equations. We believe this technique is a powerful tool, and in the future could be used in other settings as well.

3.1 Protocol I: Semi-honest Receiver and Malicious Sender

In this single-round protocol, we consider a semi-honest receiver and a malicious sender. There are three action phases in the protocol as given below.

Step 1. Receiver's Request: The receiver creates a public-private key pair (k_{pu}, k_{pr}) in probabilistic public key encryption scheme. While the private key k_{pr} is kept confidential with the receiver, the public key (k_{pu}) is revealed to the sender. We denote the number of bits to represent the keys as l (i.e., $|k_{pu}| = l$ and $k_{pu} < 2^l$). The receiver creates a n-bit selection string X as explained before, where bit $X_i = 1$ if the i^{th} data value is to be retrieved from the sender. Otherwise, X_i is set to 0. Thus, k out of n bits of the string X are set to 1. To hide the selections from the sender, X is encrypted bit-wise using the public key k_{pu}: $\bar{X} = \{E(X_i), \ldots, E(X_n)\}$. The receiver sends the encrypted bits $\bar{X}_1, \ldots, \bar{X}_n$ to the sender.

Step 2. Sender's Reply: The data values $v_1, \ldots, v_n$ are stored at the sender. We denote the number of bits to represent any v_i as α (i.e., $v_i < 2^\alpha$). The sender receives $\bar{X}$ but cannot distinguish between $X_i = 1$ and $X_i = 0$ due to the semantic security of the encryption scheme. The sender creates $\bar{Z}_1, \ldots, \bar{Z}_n$ from $\bar{X}$ and $v_1, \ldots, v_n$ as $\bar{Z}_i = \bar{X}_i \times_h v_i$, where $\times_h$ is the multiplication of constant against a ciphertext, allowed in the encryption scheme. Note that $\bar{Z}_1, \ldots, \bar{Z}_n$ are in encrypted form, and consequently the sender has no way of knowing which k values are selected by the receiver. The sender sends $\bar{Z}_1, \ldots, \bar{Z}_n$ to the receiver.

Step 3. Unpacking at Receiver: The receiver decrypts the values of $\bar{Z}_1, \ldots, \bar{Z}_n$ to get $Z_1, \ldots, Z_n$ When $X_i = 1$, the corresponding Z_i is v_i. When $X_i = 0$, the corresponding $\bar{Z}_i$ is $E(0)$, and thus the receiver gets no information about the data values not selected by X. These values reveal the k data values for the indices when $X_i = 1$.

Security Analysis: We now analyze why this protocol is secure in the presense of a semi-honest receiver and a malicious sender. There are two possible effective malicious behaviors for the sender: 1) To discover receiver's selection of k values, and 2) To input the wrong data values.

The sender receives the following from the receiver: $\bar{X}_1, \ldots, \bar{X}_n$, k and k_{pu}. While k is a public knowledge, k_{pu} is a random value to the sender. The n ciphertexts, $\bar{X}_1, \ldots, \bar{X}_n$ are generated using the key k_{pu} by the receiver. These are from a semantically secure encryption scheme. Let $C_1^*, \ldots, C_n^*$ be a set of randomly generated numbers containing same number of bits as $\bar{X}_i$s. Since the encryption scheme is semantically secure and the sender's computing power is polynomially bounded, it is impossible for the sender to distinguish between $\bar{X}_1, \ldots, \bar{X}_n$ and $C_1^*, \ldots, C_n^*$ without knowing k_{pr}. As a result, the sender will not gain any non-negligible knowledge regarding the index values from $\bar{X}_1, \ldots, \bar{X}_n$.

Under the second malicious behavior, the sender refuses to properly execute the steps of the protocol, resulting in receiver getting wrong results. This is equivalent to input modification problem, where the sender purposefully modifies the data values. Zero-knowledge proofs [10] can be used to prevent the sender from changing data values once they are committed. However, that does not prevent the sender from committing wrong values to start with. Thus even zero-knowledge proofs cannot prevent such behavior. We consider the problem of input modification to be outside the scope of this problem.

A semi-honest receiver always sends a well-formed selection string X to the sender. Suppose the receiver tries to decrypt more than k values from $\bar{Z}_i, \ldots, \bar{Z}_n$ to obtain the corresponding data values. Since the server has multiplied the selection bit with the data values, only k of $\bar{Z}_i, \ldots, \bar{Z}_n$ will be non-zeros, and the remaining $\bar{Z}_i$ will contain 0. Thus the receiver will not gain any additional knowledge about the non-selected data values.

Complexity: We define the computational complexity as the number of encryptions/decryptions performed, since they are the most expensive operation in the protocol. To encrypt the selection string, the receiver performs n encryptions, and to get the results at the end, it performs n decryptions. At the sender's side, there are n homomorphic multiplications. Thus the sender's complexity is bounded by $\mathrm{O}(n)$. The overall computational complexity of the protocol is $\mathrm{O}(n)$.

Communication complexity is measured as the number of bits exchanged between the server and the receiver. The receiver transfers the selection string as n ciphertexts. At the final step, the sender sends back n ciphertexts. As a result, the overall communication complexity of the protocol is $\mathrm{O}(l \cdot n)$ in bits, where l is the number of bits of the encryption key.

3.2 Protocol II : Malicious Receiver and Malicious Sender

Protocol I is secure when the receiver is semi-honest, but fails when we consider a malicious receiver. If a receiver sets more than k bits in X to 1, it will reveal more than k data values to the receiver.

We now present a variation of Protocol I that works even in the presence of malicious receiver and malicious sender. Instead of sending only one selection string, in Protocol II the receiver sends m encrypted selection strings. Out of these m strings, $m-1$ strings are revealed to the sender, and are verified by the sender to be well-formed. These strings are formed such that the opened selection strings do not reveal the actual user selection. The sender uses the remaining un-opened selection string for OT. This protocol prevents the receiver from getting more than k values with $\frac{m-1}{m}$ probability. Moreover, the receiver is caught if he tries to cheat. We now explain the steps of Protocol II in detail. The corresponding algorithm is given in Algorithm 1.

Step 1. Receiver's Request: The receiver first creates m public-private key pairs (k_{pu}^j, k_{pr}^j). It also generates random selection strings with n-bits each such that only k random bits are set to 1 and the remaining $n-k$ bits are set to 0 (i.e., well-formed w.r.t. n and k). There are $\frac{n!}{k!(n-k)!}$ possible n-bit strings with k 1's and $n-k$ 0's. When $n = 20$ and $k = 10$, the number of possible strings is $184,756$. The receiver generates m such strings. Let these strings be $X^1, \ldots, X^m$, and we denote the i-th bit of string X^j as X_i^j. Note that these strings contain 1's in random positions and the ordering has no relation to the indices of the data values selected by the receiver. Let X^{real} be the actual selection string intended by the receiver. This string is kept secret at the receiver. Using the m public keys, the receiver encrypts the m random strings as follows:

$$\bar{X}^1 = E_1(X_1^1), \ldots, E_1(X_n^1)$$
$$\vdots$$
$$\bar{X}^m = E_m(X_1^m), \ldots, E_m(X_n^m)$$

E_j is the encryption using key k_{pu}^j. The receiver sends $\bar{X}^1, \ldots, \bar{X}^m$ to the sender, along with the individual cryptographic hashes (e.g., SHA-1, SHA-2) of the m private keys (step 1(e) of Algorithm 1).

Example 1. For $n = 5$ and $k = 2$, let $X^{real} = \{0, 1, 0, 1, 0\}$ be the actual user selection string. For $m = 2$, let us assume the receiver generates two random strings as $X^1 = \{1, 0, 0, 1, 0\}$ and $X^2 = \{0, 0, 1, 1, 0\}$.

Step 2. Sender's Reply: The sender randomly picks $m-1$ encrypted strings from $\bar{X}^1, \ldots, \bar{X}^m$ and requests their encryption keys (step 2(a)) so that these encrypted strings can be decrypted and verified to be well-formed (to contain exactly k 1's and $n-k$ 0's).

Step 3. Receiver's reply: The receiver gets a request for $m-1$ private keys from the sender. By sending these $m-1$ private keys the receiver can now prove to the sender that $m-1$ (out of m) selection strings are well-formed. Let us assume u $(1 \leq u \leq m)$ is the index of the only one selection string whose private key is not requested. This means that $\bar{X}^u = \bar{X}_1^u, \ldots, \bar{X}_n^u$ is the selection string that remains secret.

While $\bar{X}^u$ is not opened by the sender and subsequently used for the oblivious transfer, $\bar{X}^u$ is a random string that is well-formed for k. X^u contains k 1's and $n-k$ 0's but is not necessarily the equivalent of X^{real}, the actual selection string intended by the receiver. To make X^u and X^{real} equivalent, the receiver finds a permutation P that permutes the sequence $\bar{X}^u$ to get $\bar{X'}^u$ so that X'^u and X^{real} are equivalent. $\bar{X'}^u$ contains the same ciphertexts of $\bar{X}^u$ but re-arranged so that X^{real} and X'^u are the same, i.e., both retrieve same data values from the sender. Since the $\bar{X'}^u$ is still in encrypted form, the sender cannot get the real selection string. The receiver also sends the public key of u (k^u_{pu}) to the sender (steps 3(a)-(c)).

Example 2. From Example 1, let us assume that the sender wants to open $\bar{X}^1$. The selection string X^2=$\{0, 0, 1, 1, 0\}$ remains un-opened, and the sender has the encrypted $\bar{X}^2$ as $\{E_2(0), E_2(0), E_2(1), E_2(1), E_2(0)\}$, Let these be $\{e_1, \ldots, e_n\}$. The real user selection string X^{real} is $\{0, 1, 0, 1, 0\}$. The receiver now computes a permutation P so that entries of $\bar{X}^2$ will give X^{real}. One such possible permutation is $\{1 \rightarrow 3, 2 \rightarrow 5, 3 \rightarrow 2, 4 \rightarrow 4, 5 \rightarrow 1\}$ that produces $\bar{X'}^2$ as $\{e_5, e_3, e_1, e_4, e_2\}$, which is equivalent to $\{E_2(0), E_2(1)$,$E_2(0), E_2(1)$,$E_2(0)\}$. There are also other permutations that will result in X^{real}.

4. Sender's reply: After receiving the $m-1$ keys, the sender first verifies whether the hash values of the keys match the hash values sent by the receiver at Step 1(e). If any of them does not match, the sender stops the protocol as the receiver is attempting to send different keys than the ones used at Step 1(d). If the hash values match, the sender then verifies whether all $m-1$ decrypted selection strings are well-formed. If any of the strings is not well-formed, the sender stops the protocol as the receiver is attempting to get more than k values. If all the $m-1$ strings are well-formed, the sender is assured with $\frac{1}{m}$ probability that the un-opened string is also well-formed.

The sender uses the permutation P to re-order the un-opened string $\bar{X}^u$ to get a new sequence $\bar{X'}^u$. It computes $\bar{Z}_1, \ldots, \bar{Z}_n$ from $\bar{X'}^u$ and $v_1, \ldots, v_n$, where $\bar{Z}_i = \bar{X'}^u_i \times_h v_i$. The sender sends $\bar{Z}$ values to the receiver.

3. Unpacking at Receiver: The receiver decrypts the values of $\bar{Z}_1, \ldots, \bar{Z}_n$ to get the k data values for the indices when $X^{real}_i = 1$ (step 5(a)).

Security Analysis: The main difference between Protocol I and II is that the receiver could act malicious in Protocol II. Suppose the receiver is dishonest, then the only possible malicious behavior is to obtain more than k values from the sender. (Though it is possible for the receiver to obtain less than k values, retrieving less than k values is not considered as an attack on the protocol.)

At step 4(d) of Algorithm 1, the sender computes the component-wise product of $\bar{X'}^u$ with the data values. To get more than k data values, it is imperative that $\bar{X'}^u$ must contain more than k non-zero values. To achieve this, the receiver needs to send more than k encryptions of non-zero values in one of the selection strings from $X^1, \ldots, X^m$ at Step 1(b). However, $\bar{X'}^u$ is used at step 4(d) only if all the remaining $m-1$ selection strings are opened by the sender at step 4(b). For the receiver to succeed in cheating, the selection string that contains more than k non-zero values should not be selected at step 2(a) by the sender. This occurs with probability $\frac{1}{m}$. Moreover, this selection string is selected with $\frac{m-1}{m}$

Algorithm 1 Protocol II: k-out-of-n Oblivious Transfer

```
Require: Receiver's inputs: $X_1, \ldots, X_n$, $k$; Sender's inputs: $v_1, \ldots, v_n$.
Receiver:
    (a). Create $n$-bit selection string as $X^{real}$; Set $X_i^{real} = 1$ to retrieve value $v_i$; otherwise, set $X_i^{real} = 0$.
    (b). Create $m$ random $n$-bit strings, $X^1, \ldots, X^m$, so that they contain exactly $k$ 1's and $n - k$ 0's.
    (c). Create $m$ public-private key pairs $(k_{pu}^1, k_{pr}^1)$,...,$(k_{pu}^m, k_{pr}^m)$.
    (d). for $j = 1$ to $m$ do
        (d.1). Compute $\bar{X}_i^j \leftarrow E_j(X_i^j)$ for $i = 1, \ldots, n$.
        (d.2). Compute $h_j \leftarrow HASH(k_{pr}^j)$.
    (e). Send $\bar{X}^1, \ldots, \bar{X}^m$, $k$, $h_1, \ldots, h_m$ to the sender.
Sender:
    (a). Randomly select $m - 1$ selection strings and request the private keys from receiver.
Receiver:
    (a). Let $\bar{X}^u$ be the selection string that is not selected by the sender.
    (b). Compute a permutation $P$ that makes $\bar{X}^u$ and $X^{real}$ equivalent.
    (c). Send the permutation $P$, $m - 1$ private keys requested by the sender and the public key $k_{pu}^u$ of $\bar{X}^u$.
Sender:
    (a). Verify whether the hash values of the $m - 1$ keys match with $h_1, \ldots, h_m$. If not, stop the protocol.
    (b). Decrypt the $m - 1$ selection strings with their corresponding keys sent by the receiver. Verify whether all the $m - 1$ strings are well-formed. If not, stop the protocol.
    (c). Use the permutation $P$ to compute $\bar{X}'^u$ from the only remaining selection string, $\bar{X}^u$.
    (d). Compute $\bar{Z}_i = \bar{X}'^u_i \times_h v_i$, for $i = 1, \ldots, n$.
    (e). Send $\bar{Z}_i, \ldots, \bar{Z}_n$ to the receiver.
Receiver:
    (a). Retrieve $k$ data values by computing $v_i = D(\bar{Z}_i)$ for all $i$ when $X_i^{real} = 1$.
```

probability, and if that happens, the receiver is caught cheating by the sender at step 4(b).

Let us suppose a cheating receiver sends a selection string $\bar{X}'$ with more than k 1's to the sender at Step 1(e), and this string is selected by the sender at Step 2(a). At step 3(c), the receiver may try to send a different key (k'') than the one used (k') in the encryption of $\bar{X}'$. However, to go undetected by the sender at step 4(b), the receiver needs to find a key k'' such that the hashes of k' and k'' are the same. This is equivalent to finding a collision in hash functions, which is extremely unlikely for SHA-1 or SHA-2. Thus the receiver cannot change the keys once the sender requests $m-1$ keys.

A malicious sender may try to learn the selections of the receiver. At step 1(e), the sender receives the encryptions of m random well-formed strings as $\bar{X}^1, \ldots, \bar{X}^m$, along with the hashes $h_1, \ldots, h_m$ of the private keys. Only one of these selection strings remains un-opened at step 4(b). So, the other $m-1$ sequences are random strings generated by the receiver with k 1's and $n-k$ 0's. As there are $\frac{n!}{k!(n-k)!}$ such combinations, these random strings do not reveal any information about X^{real}, the actual user selection. The sender itself could have generated such strings. The hash values are generated from one-way cryptographic hash functions. Thus, computing the actual strings (i.e., randomly generated private keys) from hashes $h_1, \ldots, h_m$, involve brute force attack on the key space (in the range of 2^{1024}), which is computationally impossible.

Let us now consider the selection string $\bar{X}^u$ that is not opened by the sender. This contains n ciphertexts, encryptions of 1's and 0's as $\{\bar{X}^u_1, \ldots, \bar{X}^u_n\}$. The sender learns nothing about the contents of these ciphertexts due to semantic-security property of the encryption scheme. At step 4(c), the receiver uses the permutation P on $\bar{X}^u$ to get a new sequence, $\bar{X}'^u_1, \ldots, \bar{X}'^u_n$. Since the sender gains no knowledge about user's selection from $\bar{X}^u$, the re-ordered selection string $\bar{X}'^u$ also does not reveal any information. Thus a malicious sender will not gain any information about the receiver's selection in this protocol.

Complexity: At step 1(d) of Algorithm 1, the receiver performs $n \cdot m$ encryptions, and n decryptions at step 5(a). At the sender's side, there are $n \cdot (m-1)$ decryptions at step 4(b), and n homomorphic multiplications at step 4(d). Thus the overall computational complexity of the protocol is $\mathrm{O}(n \cdot m)$.

At step 1(e), the receiver sends $n \cdot m$ encrypted values and m hash values. Since the ciphertexts have more bits than the hashes, the communication complexity is bounded by $\mathrm{O}(l \cdot n \cdot m)$ bits, where l is the number of bits of the encryption key. The sender sends back n ciphertexts to the receiver at step 1(e). As a result, the overall communication complexity of the protocol is $\mathrm{O}(l \cdot n \cdot m)$ in bits. Protocol II needs an extra interaction between the sender and receiver as compared to Protocol I and III.

3.3 Protocol III - Hybrid Scheme

We now present a hybrid (based on cryptography and algebra) protocol for OT^n_k. This scheme allows the receiver to construct k equations with k unknowns, where the unknowns are the data values from the sender. By solving the equations, the receiver gets the k data values it wants to retrieve. The equations are hidden (i.e., in encrypted form) from the sender such that the selected values are kept confidential from the sender. If the receiver attempts to retrieve more than k values, then the receiver is unable to even form the equations. (See Appendix 6.3 for a general discussion on the solvability of linear equations). Thus the greedy receivers will not get any data value at all. Algorithm 2 lists the steps of the protocol. We now describe each phase of the protocol in detail.

Step 1. Receiver's Request: Similar to Protocol I, the receiver creates a n-bit selection string X where bit $X_i = 1$ if the i-th value is to be retrieved from the sender. Otherwise, X_i is set to 0. private key k_{pr}: $\bar{X}_i = E(X_i)$. The receiver sends the encrypted bits $\bar{X}_1, \ldots, \bar{X}_n$ to the sender.

Example 3. Refer to Table 1 that shows an example OT_2^5. As shown in Table 1(a), the receiver sets X_2 and X_4 to 1 so that v_2 (35) and v_4 (85) are retrieved.

Step 2. Sender's Reply: The sender generates two $k \times n$ random perturbations matrices, A and B. The matrix A is generated using a random number R_1^1 as the seed to the random number generator. Thus A consists of $k \cdot n$ random positive integer values. We denote the number of bits to represent any $A_{i,j}$ as β (i.e, $A_{i,j} < 2^\beta$). A can be completely re-generated given only the value of R_1^1.

The matrix B is generated by using n random seeds $R_1^2, \ldots, R_n^2$. Each random seed (R_i^2) is used in generating k numbers which form the i-th column in matrix B. Any i-th column of B can be completely re-generated given only the value of R_i^2. We also require that all the columns and rows are independent in A and B, i.e., their rank is k (assuming $k \leq n$). This property ensures that the resultant k equations are independent, which are then solved by the receiver. A randomly constructed matrix usually has this property. While A is revealed to the receiver (through R_1^1), B is kept secret at the sender.

In the first step, the sender uses A and $R_1^2, \ldots, R_n^2$ to compute a $k \times n$ matrix Y^1, where $Y_{i,j}^1 = R_j^2 * A_{i,j}$ for $i = 1, \ldots, k$ and $j = 1, \ldots, n$ (step 2(c)).

$$Y^1 = \begin{pmatrix} R_1^2 * A_{1,1} \ , \ldots , \ R_n^2 * A_{1,n} \\ \cdots \\ R_1^2 * A_{k,1} \ , \ldots , \ R_n^2 * A_{k,n} \end{pmatrix}$$

Using Y^1, the sender computes $\bar{Z}_1^1, \ldots, \bar{Z}_k^1$ as follows (step 2(e)):

$$\bar{Z}_1^1 = \bar{X}_1 \times_h Y_{1,1}^1 +_h \ldots +_h \bar{X}_n \times_h Y_{1,n}^1$$
$$\vdots$$
$$\bar{Z}_k^1 = \bar{X}_1 \times_h Y_{k,1}^1 +_h \ldots +_h \bar{X}_n \times_h Y_{k,n}^1$$

$+_h$ is the homomorphic addition operation. Note that $\bar{Z}_1, \ldots, \bar{Z}_k$ are in encrypted form and thus the sender has no way of knowing which k values are selected by the receiver. These k values and the matrix A can be used by the receiver to construct k equations with k unknowns (assuming X is well-formed w.r.t n and k), where the unknowns are k of the n random seed values, $R^2_1, \ldots, R^2_n$.

In the second step, the sender uses the perturbation matrix B and $v_1, \ldots, v_n$ to compute a $k \times n$ matrix Y^2, where $Y^2_{i,j} = v_j * B_{i,j}$ for $i = 1, \ldots, k$ and $j = 1, \ldots, n$ (step 2(d)).

$$Y^2 = \begin{pmatrix} v_1 * B_{1,1} \;, \ldots, \; v_n * B_{1,n} \\ \ldots \\ v_1 * B_{k,1} \;, \ldots, \; v_n * B_{k,n} \end{pmatrix}$$

i	X_i	$\bar{X}_i$	v_i
1	0	E(0)	50
2	1	E(1)	35
3	0	E(0)	46
4	1	E(1)	82
5	0	E(0)	28

(a) Inputs

$$\text{A=}\begin{pmatrix} 2 & 3 & 7 & 6 & 1 \\ 10 & 4 & 2 & 7 & 9 \end{pmatrix}$$

$$R^2 = \{12, 3, 7, 2, 4\}$$

$$\text{B=}\begin{pmatrix} 4 & 6 & 1 & 3 & 9 \\ 5 & 11 & 2 & 2 & 4 \end{pmatrix}$$

(b) Random Matrices: $A_{2\times5}$ and $B_{2\times5}$

$$Y^1\text{=}\begin{pmatrix} 24 & 9 & 49 & 12 & 4 \\ 120 & 12 & 14 & 14 & 36 \end{pmatrix}$$

$$Y^2\text{=}\begin{pmatrix} 200 & 210 & 46 & 246 & 252 \\ 250 & 385 & 92 & 164 & 112 \end{pmatrix}$$

(c) Matrices Y^1 and Y^2

$$Z^1_1 = 9 + 12 = 21$$
$$Z^1_2 = 12 + 14 = 26$$

$$Z^2_1 = 210 + 246 = 456$$
$$Z^2_2 = 385 + 164 = 549$$

(d) Computing Z^1 and Z^2

$$3r_2 + 6r_4 = 21$$
$$4r_2 + 7r_4 = 26$$

Solve to get
$r_2 = 3$, $r_4 = 2$

3 generates $\{6, 11\}$
2 generates $\{3, 2\}$

(e) Solving for Random seeds

$$\text{B=}\begin{pmatrix} * & 6 & * & 3 & * \\ * & 11 & * & 2 & * \end{pmatrix}$$

$$6v_2 + 3v_4 = 456$$
$$11v_2 + 2v_4 = 549$$

Solve to get
$v_2 = 35$, $v_4 = 82$

(f) Final steps in OT$^5_2$

Table 1. An Illustration OT$^5_2$ in Protocol III

Using Y^2, the sender creates $\bar{Z}^2_1, \ldots, \bar{Z}^2_k$ as follows (step 2(f)):

$$\bar{Z}^2_1 = \bar{X}_1 \times_h Y^2_{1,1} +_h \ldots +_h \bar{X}_n \times_h Y^2_{1,n}$$
$$\vdots$$
$$\bar{Z}^2_k = \bar{X}_1 \times_h Y^2_{k,1} +_h \ldots +_h \bar{X}_n \times_h Y^2_{k,n}$$

The sender sends $\bar{Z}_1^1, \ldots, \bar{Z}_k^1$, $\bar{Z}_1^2, \ldots, \bar{Z}_k^2$, and also the random seed R_1^1 (i.e., matrix A) to the receiver.

Example 4. In Table 1(b), we show a perturbation matrix A, the secret matrix B and the random seeds for B as R^2. Table 1(c) shows matrices Y^1 and Y^2. $\bar{Z}^1$ is computed as follows: $\bar{Z}_1^1 = E(0) \times_h 24 +_h E(1) \times_h 9 +_h E(0) \times_h 49 +_h E(1) \times_h 12 +_h E(0) \times_h 4$ and $\bar{Z}_2^1 = E(0) \times_h 120 +_h E(1) \times_h 12 +_h E(0) \times_h 14 +_h E(1) \times_h 14 +_h E(0) \times_h 36$. This results in $\bar{Z}_1^1 = E(21)$ and $\bar{Z}_2^1 = E(26)$. Similarly, $\bar{Z}_1^2 = E(456)$ and $\bar{Z}_2^2 = E(549)$ are computed and sent to the receiver. □

Step 3. Unpacking at Receiver: The receiver decrypts the values of $\bar{Z}_1^1, \ldots, \bar{Z}_k^1$ to get $Z_1^1, \ldots, Z_k^1$. The decrypted values form k equations with k unknowns as follows.

$$\begin{aligned} Z_1^1 &= X_1 * R_1^2 * A_{1,1} + \ldots + X_n * R_n^2 * A_{1,n} \\ &\vdots \\ Z_k^1 &= X_1 * R_1^2 * A_{k,1} + \ldots + X_n * R_n^2 * A_{k,n} \end{aligned} \tag{1}$$

Since the receiver can re-generate the perturbation matrix A (constructed by using R_1^1), the random values $A_{i,j}$ are already known to the receiver. Assuming that X contains k 1s and $n-k$ 0s, the k equations from equation 1 will contain k unknowns which are the R_i^2's. By solving these k equations, the receiver gets the k random seeds from $R_1^2, \ldots, R_n^2$, for the columns selected by X.

Similarly, the receiver decrypts the values of $\bar{Z}_1^2, \ldots, \bar{Z}_k^2$ to get $Z_1^2, \ldots, Z_k^2$ and forms the following k equations.

$$\begin{aligned} Z_1^2 &= X_1 * v_{i_1} * B_{1,1} + \ldots + X_n * v_{i_k} * B_{1,n} \\ &\vdots \\ Z_k^2 &= X_1 * v_{i_1} * B_{k,1} + \ldots + X_n * v_{i_k} * B_{k,n} \end{aligned} \tag{2}$$

These k equations contain k unknowns which are the data values, and the co-efficients are matrix B elements from k columns. These co-efficients are known to the receiver from the previous step, by solving equation 1. Thus the receiver is able to solve equation 2 to obtain the k data values. If the protocol is run multiple times, the sender generates new perturbation matrices for each run.

Example 5. As shown in Table 1(e), the receiver forms two equations, $3r_2 + 6r_4 = 21$ and $4r_2 + 7r_4 = 26$ after receiving $E(21)$ and $E(26)$ from the sender. Note that the co-efficients are directly from the perturbation matrix A. Since the receiver knows that $X_2 = 1$ and $X_4 = 1$, it uses only the 2nd and 4th column elements from A. The solution for these equations is $r_2 = 3$ and $r_4 = 2$. This matches with R^2 in Table 1(b). The receiver generates the elements of 2nd and 4th columns of B using 3 (r_2) and 2 (r_4) and forms another set of equations, $6v_2 + 3v_4 = 456$ and $11v_2 + 2v_4 = 549$ with data values v_2 and v_4 as unknowns. The receiver solves these equations to get the data values as $v_2 = 35$ and $v_4 = 82$. □

Algorithm 2 k-out-of-n Oblivious Transfer

```
Require: Receiver's inputs: $X_1, \ldots, X_n$, $k$, $k_{pu}$, $k_{pr}$ ; Sender's inputs: $v_1, \ldots, v_n$, $R_1^1$, $R_1^2, \ldots, R_n^2$.
1: Receiver:
    (a). Set $X_i = 1$ to retrieve value $v_i$; otherwise, set $X_i = 0$
    (b). Encrypt each $X_i$ separately as $\bar{X}_i = E(X_i)$, for $i = 1, \ldots, n$; Send $\bar{X}_1, \ldots, \bar{X}_n$, $k$, $k_{pu}$ to the sender
2: Sender:
    (a). Generate the $k \times n$ perturbation matrix $A$ through $R_1^1$
    (b). Generate the $k \times n$ perturbation matrix $B$, whose elements of $i$-th column are generated by $R_i^2$.
    (c). Compute $k \times n$ matrix $Y^1$: $Y_{i,j}^1 = R_j^2 * A_{i,j}$, for $i = 1, \ldots, k$ and $j = 1, \ldots, n$
    (d). Compute $k \times n$ matrix $Y^2$: $Y_{i,j}^2 = v_j * B_{i,j}$, for $i = 1, \ldots, k$ and $j = 1, \ldots, n$
    (e). Compute $\bar{Z}_i^1 = \bar{X}_i \times_h Y_{i,1}^1 +_h \ldots +_h \bar{X}_i \times_h Y_{i,n}^1$, for $i = 1, \ldots, k$
    (f). Compute $\bar{Z}_i^2 = \bar{X}_i \times_h Y_{i,1}^2 +_h \ldots +_h \bar{X}_i \times_h Y_{i,n}^2$, for $i = 1, \ldots, k$
    (g). Send $\bar{Z}_1^1, \ldots, \bar{Z}_k^1, \bar{Z}_1^2, \ldots, \bar{Z}_k^2$ and $R_1^1$ to the receiver.
3: Receiver:
    (a). Decrypt $Z_i^1 = D(\bar{Z}_i^1)$ and $Z_i^2 = D(\bar{Z}_i^2)$ for $i = 1, \ldots, k$
    (b). Generate the perturbation matrix $A$ using $R_1^1$
    (c). Construct $k$ equations with $X$,$Z^1$ and $A$; solve the equations to get $k$ random seeds for matrix $B$
    (d). Construct $k$ equations with $X$,$Z^2$ and $k$ random seeds from 3(c); solve the equations to get the $k$ data values.
```

Security Analysis: Suppose the sender is dishonest and tries to discover receiver's selections. The only information the sender receives is the encrypted selection string, $\bar{X}_1, \ldots, \bar{X}_n$ at step 1(b). Since the encryption scheme is semantically secure and the sender's computing power is polynomially bounded, it is impossible for the sender to gain any knowledge from $\bar{X}$.

Suppose the receiver is dishonest and tries to obtain more than k values from the sender. To achieve this, the receiver first needs to send more than k encryptions of non-zero values at step 1(b). This will result in k equations (computed at steps 2(e)-(f)) with more than k unknowns. Let us consider the equations involving the random seeds at step 2(e), which the receiver solves at step 3(c). Since there are more unknowns than the equations, this will result in non-unique solutions for the random seeds. Thus the receiver will not be able to construct the equations for the data values at step 3(d) since only the unique random seeds can generate the co-efficients. Without the co-efficients, the k values $Z_1^2, \ldots, Z_k^2$ are just random values to the sender. Let us consider a scenario where the receiver generates all possible random seeds and tries to generate the co-efficients for the equation 2. Since X contains more than k 1's, for each co-efficient assigment, the receiver needs to solve k equations with more than

k unknowns by performing $\mathrm{O}(k^3)$ multiplications. For each case, there will be many non-unique solutions for the data values from equation 2. Thus a malicious receiver gets no data values at all if it attempts to cheat.

Example 6. In Table 1(a), suppose the receiver wants to retrieve v_1 also in OT_2^5. This will result in the following assignment of X and $\bar{X}$: $X = \{1, 1, 0, 1, 0\}$ and $\bar{X} = \{E(1), E(1), E(0), E(1), E(0)\}$. The sender will generate only two equations as follows: $\bar{Z}_1^1 = E(1) \times_h 24 +_h E(1) \times_h 9 +_h E(0) \times_h 49 +_h E(1) \times_h 12 +_h E(0) \times_h 4$ and $\bar{Z}_2^1 = E(1) \times_h 120 +_h E(1) \times_h 12 +_h E(0) \times_h 14 +_h E(1) \times_h 14 +_h E(0) \times_h 36$. The corresponding equations are: $2r_1 + 3r_2 + 6r_4 = 45$ and $10r_1 + 4r_2 + 7r_4 = 146$. With three unknowns, the receiver will be unable to solve these equations to get the unique values in R^2 to generate the columns of B. This means that the equations for the data values cannot be constructed. Thus the system is secure even in the presence of greedy receivers. □

Complexity Analysis: At step 1(b) of Algorithm 2, the receiver performs n encryptions, and $2k$ decryptions at step 3(a). Also, at steps 3(c)-(d), the receiver solves k equations which results in a complexity of $\mathrm{O}(k^3)$ ordinary multiplications. However, the expensive encryptions dominate the $\mathrm{O}(k^3)$ operations to result in $\mathrm{O}(n)$ as the complexity for the receiver. At the sender's side, there are $n \cdot k$ homomorphic additions and $n \cdot k$ multiplications at steps 2(e)-(f). Thus the computational complexity of the protocol is $\mathrm{O}(n \cdot k)$.

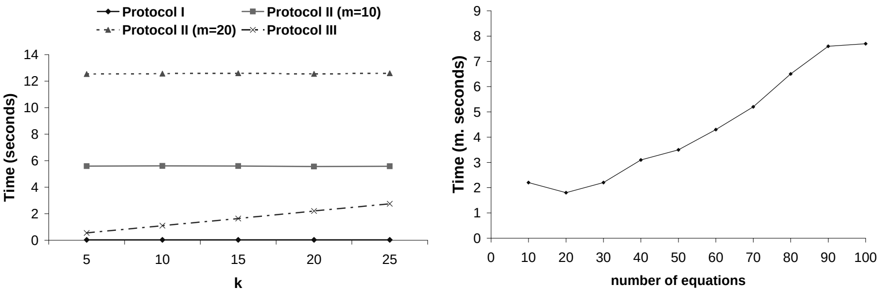
At step 1(b), the receiver sends n encrypted values. At step 2(g), the sender sends back $2k$ ciphertexts to the receiver. Since $n > k$, the overall the communication complexity of the protocol is $\mathrm{O}(l \cdot n)$ in bits.

4 Experimental Analysis

We now report the performance of the protocols, implemented in C and executed on Ubuntu 8.10 with a Intel dual core 2.33 GHz processor and 4 GB RAM. Since computational complexity is the major bottleneck, we show the performance results in terms of time taken at the sender side, since the receiver's computation is a constant number of decryptions. The sender's data values are generated randomly, with the maximum being 2^{32} (integer values). Similarly, the random matrix elements are also random integers. We use Paillier encryption scheme for all the three protocols.

Figure 1(a) shows the effect of k on the running time for a fixed $n = 100$. For this experiment, the key size in the Paillier encryption is set to 1024 bits, sufficiently large for practice (see Appendix 6.4 for a discussion on selecting key size based on the size of data values). The Y-axis shows the running time in seconds for k ranging from 5 to 25, while n is fixed at 100. Protocol I is the most efficient, taking a constant time of .03 seconds. Protocol II with $m = 10$ and $m = 20$ take 6 and 12.5 seconds respectively. Protocol III's running time increases sub-linearly as k increases, but remains well under Protocol II ($m = 20$).

At step 3(c)-(d) of Algorithm 2 (Protocol III), the receiver solves equations to get the k values from the sender. In this experiment, we measure the time



(a) OT_k^{100}: Time with varying k (b) Time taken for solving equations

Fig. 1. Experimental Results

taken for solving the equations. We used the Matlab software to solve the linear systems by varying the number of equations, which is kept same as the nuAmber of unknowns. Figure 1(b) shows how the running time changes sub-linearly as the number of equations increases. For solving 50 equations (with 50 unknowns) it takes approximately 3.5 milli seconds, and for 100 equations the running time is 7.7 milli seconds. Compared to the time spent on encryptions and decryptions from Figure 1(a), the time taken for this step is negligible.

5 Conclusion and Future work

In this paper, we present three efficient k-out-of-n OT protocols that take advantage of additive homomorphic crypto system. While Protocol I is efficient, it does not protect the sender from malicious receiver. Protocol II is efficient, and withstands malicious behaviors from sender and receiver. Moreover, the receiver is caught if it attempts to retrieve more than k values with $1/m$ probability. Protocol III presents a novel technique based on the solvability of linear equations. A malicious receiver will be unable to solve k equations with more than k unknowns. Protocols I and III are very efficient in terms of communication cost since they require only one round and exchange of $\mathrm{O}(n)$ messages from the receiver to the sender. Protocol II require one additional interaction, which we plan to eliminate in our future work. Furthermore, our protocols are efficient in terms of computation complexity as shown in the experimental results.

As a future work, we shall improve the complexity of communication from the receiver to sender. In case of retrieving small number of items obliviously from a large database, there will be a big network overhead while sending the encrypted index values to the sender. A more efficient protocol could be less dependent on the size of the database.

References

1. J. C. Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret. In A. Odlyzko, editor, *Advances in Cryptography, CRYPTO86: Proceedings*, volume 263, pages 251–260. Springer-Verlag, Lecture Notes in Computer Science, 1986.
2. M. Blum. Three application of oblivious transfer: Part i: Coin flipping by telephone; part ii: How to exchange secrets; part iii: How to send certified electronic mail. 1981.
3. G. Brassard, C. Crépeau, and J.-M. Robert. Information theoretic reductions among disclosure problems. In *FOCS*, pages 168–173, 1986.
4. G. Brassard, C. Crépeau, and J.-M. Robert. All-or-nothing disclosure of secrets. In *Proceedings on Advances in cryptology—CRYPTO '86*, pages 234–238, London, UK, 1987. Springer-Verlag.
5. C.-K. Chu and W.-G. Tzeng. Efficient k-out-of-n oblivious transfer schemes. *Journal of Universal Computer Science*, 14(3):397–415, feb 2008.
6. C. Crépeau. Equivalence between two flavours of oblivious transfers. In *CRYPTO '87: A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, pages 350–354, London, UK, 1988. Springer-Verlag.
7. C. Crépeau and J. Kilian. Weakening security assumptions and oblivious transfer. In *CRYPTO '88: Proceedings on Advances in cryptology*, pages 2–7, New York, NY, USA, 1990. Springer-Verlag New York, Inc.
8. I. Damgard, M. Jurik, and J. Nielsen. A generalization of paillier's public-key system with applications to electronic voting, 2003.
9. O. Goldreich. *The Foundations of Cryptography*, volume 2, chapter General Cryptographic Protocols. Cambridge University Press, 2004.
10. O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *Journal of ACM*, 38:690–728, 1991.
11. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.
12. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing (STOC'85)*, pages 291–304, Providence, Rhode Island, U.S.A., May 6-8 1985.
13. B. Malek and A. Miri. Optimal secure data retrieval using an oblivious transfer scheme. In *IEEE International Conference on Wireless And Mobile Computing, Networking And Communications*, volume 2, pages 25–31, 2005.
14. H. min Sun, K. hang Wang, and C. fu Hung. Towards privacy preserving digital rights management using oblivious transfer, 2006.
15. D. Naccache and J. Stern. A new public key cryptosystem based on higher residues. In *Proceedings of the 5th ACM conference on Computer and communications security*, pages 59–66, San Francisco, California, United States, 1998. ACM Press.
16. M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 245–254, New York, NY, USA, 1999. ACM.
17. T. Okamoto and S. Uchiyama. A new public-key cryptosystem as secure as factoring. In *Advances in Cryptology - Eurocrypt '98, LNCS 1403*, pages 308–318. Springer-Verlag, 1998.
18. P. Paillier. Public key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - Eurocrypt '99 Proceedings, LNCS 1592*. Springer-Verlag, 1999.

19. M. O. Rabin. How to exchange secrets by oblivious transfer. Technical report, Aiken Computation Laboratory, Harvard University, 1981.
20. M. Stanek. M.: Efficient simultaneous contract signing. In *In 19th International Conference on Information Security (SEC 2004), 18th IFIP Word Computer Congress*, pages 441–455. Kluwer Academic Publishers, 2004.
21. Q. Wu, B. Qin, C. Wang, X. Chen, and Y. Wang. t-out-of-n string/bit oblivious transfers revisited. In *ISPEC*, pages 410–421, 2005.

6 Appendix

6.1 Deficiency of MM05-OT Protocol

As we mentioned before, there is a flaw related to the MM05-OT protocol that makes it to produce incorrect results. One of the key steps in the MM05-OT protocol is to encrypt a *ciphertext* using Paillier's scheme. However, as we show below, encrypting a ciphertext under Paillier's scheme will "almost always" result in incorrect result.

The main idea behind the MM05-OT protocol is to submit the encrypted index bits of the selection to sender. For example, to select the 3rd entry out of 8 values, the receiver sends $[E(0), E(1), E(1)]$ and $[E(1), E(0), E(0)]$ to the sender. The following construction is an example commonly used in the MM05-OT protocol.

$$E(\bar{x}_1 \cdot E(\bar{x}_0 v_0 + x_0 v_1)) = E(\bar{x}_1)^{E(\bar{x}_0 v_0 + x_0 v_1)}$$

Thus the MM05-OT protocol relies on using ciphertexts in the multiplicative operation of the Paillier. However, this step "almost always" causes the MM05-OT protocol to produce incorrect results. We have the following claim:

Claim. The probability for MM05-OT to produce correct results is negligible.

Proof. Let P denote the probability that the MM05-OT protocol produces the correct result, l be the size of the encryption key in bits, and M be a plaintext According to Paillier's scheme, $E(M)$ is uniformly distributed in $\left[0, 2^{2l} - 1\right]$, for any M in $\left[0, 2^l - 1\right]$. Since the MM05-OT protocol requires encrypting a ciphertext, the protocol generates correct results only when $E(M)$ is in $\left[0, 2^l - 1\right]$. Therefore, we have the following analysis:

$$\begin{aligned} P &= Prob\left(E(M) \in \left[0, 2^l - 1\right]\right) \\ &= \frac{2^l}{\left(2^l\right)^2} = \frac{1}{2^l} \end{aligned}$$

In practice, l is generally greater than 1024. Thus, $P \leq \frac{1}{2^{1024}} \rightarrow 0$. As a consequence, the MM05-OT protocol almost never generates correct results.

6.2 Additive Homomorphic Encryption

In our hybrid OT_k^n protocol, we adopt a probabilistic public key encryption scheme with the additive homomorphic property. Now we briefly present some of its key features (assuming that all the values are in appropriate domains).

1. The encryption function is additive homomorphic, i.e., $\forall (r_1, x_1), (r_2, x_2) \in R \times X, E(r_1, x_1) +_h E(r_2, x_2) = E(r_3, x_1 + x_2)$, where r_3 can be computed from r_1, r_2, x_1 and x_2 in polynomial time. ($+_h$ indicates the operation to "add" two encrypted values).
2. The encryption function has *semantic security* as defined in [12]. Informally speaking, a set of ciphertexts do not provide additional information about the plaintext to an adversary with polynomial-bounded computing power.
3. The encryption function is probabilistic, i.e., if $r_1 \neq r_2$, then $E(r_1, x) \neq E(r_2, x)$ but $D(E(r_1, x)) = D(E(r_2, x))$, where D denote the decryption function. This property can be derived from the *semantic security* definition.
4. Given a constant k and a ciphertext $E(r_1, x)$, we can efficiently compute $k \times_h E(r_1, x) = E(r_2, k \cdot x)$ ($\times_h$ indicates the operation to multiply a ciphertext with a constant).

Note that our protocol is a generic in that any homomorphic probabilistic public key encryption systems, such as those proposed in [1, 15, 17], can be adopted in its implementation. In our empirical study (Section 4), we adopt Paillier's cryptosystem [18] since it is efficient and commonly used in practice.

6.3 Solving Linear Equations:

Since Protocol III (Section 3.3) uses the idea of solving linear equations, we now give a brief introduction to solving m equations with n unknowns. Let the system of equations be the following:

$$\begin{aligned} b_1 &= a_{11} \cdot x_1 + \cdots + a_{1n} \cdot x_n \\ &\vdots \\ b_m &= a_{m1} \cdot x_1 + \cdots + a_{mn} \cdot x_n \end{aligned}$$

The unknowns are $x_1, \ldots, x_n$, while $a_{11}, \ldots, a_{mn}$ are the coefficients and $b_1, \ldots, b_m$ are the constant values. The solution to this problem is an assignment of values to the unknowns $x_1, \ldots, x_n$. This system of equations have different solutions depending on the values of m and n:

1. $m < n$: If the number of equations is less than the number of unknowns, then there are infinitely many solutions. If we assume only positive values for the unknowns, then the solutions for each unknown x_i is in the range $(0, MIN(\frac{b_1}{a_{1i}}, \ldots, \frac{b_m}{a_{mi}}))$. An exhaustive search in this space will give all the possible assignments for the unknowns, resluting in many solutions. Thus the solution is not unique when $m < n$.
2. $m = n$: If the number of equations and unknowns are the same and the equations are linearly independent, then there is a single unique solution.

The complexity of solving n equations with n unknowns is typically O(n^3).

6.4 Selection of the Key Size (l) :

The encryption function of the paillier scheme requires that the plaintext x is less than 2^l, where l is the key size. The data values v_i's are assumed to be less than 2^α, and the perturbation matrix entries are less than 2^β. In the randomization step (step (d) of Algorithm 2), the data values are multiplied with the perturbation matrix to produce matrices, Y^2. This results in the entries of Y^2 to be less than $2^{\alpha+\beta}$. In Step 3 (Unpacking at Receiver) of the protocol, the receiver decrypts $\bar{Z}_i^2$ to get Z_i^2. Suppose $\alpha + \beta > l$, then it would result in a $v_i * B_{ji}$ that is greater than 2^l. This results in wrong solutions to be computed at the receiver. Thus, the essential condition for the protocol to succeed is $\alpha+\beta < l$. With this condition, the entries $Y_{i,j}^2$'s are less than 2^l. Based on the values of α and β, the receiver selects the key size.

Let us consider Z_1^2 as $z_1 + .. + z_k$ where $z_1 = v_{i_1} * B_{1,i_1}$. It is possible that while each z_i is less than 2^l, the sum Z_1 may be larger than 2^l. Though this is very unlikely to happen, we can alleviate this problem by selecting the key size (l) such that $\alpha + \beta + k < l$, where k is the number of data values transferred. This condition is sufficient for the protocol to succeed all the time. The protocols in Certified E-mail [2] and Simultaneous Contract Signing [20] use OT for transferring keys of symmetric key encryption schemes such as DES and AES. The generally accepted key size for AES is 128 bits (α). For random numbers in matrix A and B, 32 is a sufficient β value. This leaves the key size(l) for the Paillier scheme in our protocol to be greater than 160 bits. Thus the key size of 1024 is sufficient for the practical purposes of running our protocol.