Secure computation of functionalities based on Hamming distance and its application to computing document similarity

Ayman Jarrous

Department of Computer Sciences,
University of Haifa,
Mount Carmel, Haifa 31905, Israel
E-mail: ayman@jarrous.net

Benny Pinkas*

Department of Computer Science,
Bar Ilan University,
Ramat-Gan 52900, Israel
E-mail: benny@pinkas.net
*Corresponding author

Abstract: This paper examines secure two-party computation of functions, which depend only on the Hamming distance of the inputs of the two parties. We present efficient protocols for computing these functions. In particular, we present protocols which are secure in the sense of full simulatability against malicious adversaries. We then show applications of HDOT. These include protocols for checking similarity between documents without disclosing additional information about them (these protocols are based on algorithms of Broder et al. for computing document similarity based on the Jaccard measure). Another application is a variant of symmetric private information retrieval (SPIR), which can be used if the server's database contains N entries, at most N / log$N$ of which have individual values, and the rest are set to some default value. The receiver does not learn whether it receives an individual value or the default value. This variant of PIR is unique since it can be based on the existence of OT alone.

Keywords: secure two-party computation; document similarity; Hamming distance; HDOT.

Reference to this paper should be made as follows: Jarrous, A. and Pinkas, B. (2013) 'Secure computation of functionalities based on Hamming distance and its application to computing document similarity', *Int. J. Applied Cryptography*, Vol. 3, No. 1, pp.21–46.

Biographical notes: Ayman Jarrous is a PhD student at the University of Haifa.

Benny Pinkas is an Associate Professor at the Department of Computer Science at Bar Ilan University, Israel. He received his PhD from the Weizmann Institute of Science in 2000. Following that, he worked at the research laboratories of Intertrust Technologies and Hewlett-Packard, and was a member of the faculty at the Department of Computer Science of the University of Haifa. His research interests include cryptography, privacy, and computer and communications security.

1 Introduction

There are many known generic constructions of secure two-party and multi-party computation, most notably the seminal constructions of Yao (1986), Goldreich et al. (1987) and Ben-Or et al. (1988). The downside of generic constructions is that they are often less efficient than tailored protocols that are designed for computing specific functionalities. It is therefore important to identify functionalities that are essential for many applications, and design efficient secure constructions of these specific functionalities. This paper performs this task for a functionality denoted as 'Hamming distance-based oblivious transfer' (HDOT), for which we also demonstrate different interesting applications. In particular, we will explore the application of that functionality for computing similarity between documents.

The Hamming distance between two strings is defined as the number of characters in which they differ. We define 'HDOT', pronounced 'h-dot') as a protocol which allows two parties, a receiver $\mathcal{P}_1$ which has an input w, and a sender $\mathcal{P}_2$ which has an input w', to securely evaluate a function $f(\cdot, \cdot)$ whose output is determined only by the

Hamming distance between w and w' (denoted $d_H(w, w')$). More precisely, the output is defined in the following way: Let $|w| = |w'| = \ell$, then $\mathcal{P}_2$ must provide $\ell + 1$ additional inputs $Z_0, \ldots, Z_\ell$, and $\mathcal{P}_1$'s output is set to be Z_d where $d = d_H(w, w')$. In this work, we design secure protocols for computing HDOT in the semi-honest and malicious scenarios, and for inputs defined over both binary and arbitrary alphabets. (A semi-honest adversary is one that follows the instructions defined by the protocol, but may try to use the information that it gained during execution in order learn about the inputs of the other parties. A malicious adversary, on the other hand, may not follow the rules of the protocol and is thus more powerful than a semi-honest adversary.) The semi-honest protocols are unique in that they invoke oblivious transfer a number of times which is only logarithmic in the input length. The malicious scenario protocols are secure according to the full simulatability definition.

On the way we define and use a new class of oblivious transfer protocols, 'constrained oblivious transfer', and show an implementation of a protocol from this class in the malicious scenario.

In more detail, this paper contains the following results:

- The paper presents the notion of Hamming distance-based oblivious transfer, HDOT, and describes protocols secure against semi-honest adversaries:

  1. A protocol denoted *bin*HDOT for *binary* inputs $w, w' \in \{0, 1\}^\ell$. This protocol operates by computing $O(\ell)$ homomorphic encryptions and only $\log \ell$ invocations of 1-out-of-2 oblivious transfer.

  2. A general HDOT protocol, for $w, w' \in \Sigma^\ell$, where Σ can be arbitrary. This protocol uses *bin*HDOT as a building block.

- A *bin*HDOT protocol secure against malicious adversaries (in the stand-alone setting). The protocol uses two primitives that must also be secure against malicious adversaries: committed oblivious transfer with constant difference (COTCD), and oblivious polynomial evaluation (OPE). We give a construction for the first primitive, which is an example of a new class of OT protocols, *constrained OT*, which we define. The latter primitive is based on a construction of Hazay and Lindell (2008).

  The security of this protocol is proved according to the full simulatability notion defined in Canetti (2000). Therefore, the composition theorem of Canetti (2000) implies that the resulting protocol can be used as a building-block for more complex protocols, and security of those latter protocols can be analysed assuming that this building-block protocol is implemented by a trusted oracle (Canetti, 2000; Goldreich, 2004).[1]

- Applications of HDOT. These include several straightforward applications, such as computing the Hamming distance between strings, or transferring one of two words based on whether the two input strings are equal or not (a functionality we denote as *EQ*, for *equality-based transfer*). Another application is a variant of symmetric private information retrieval (SPIR) which we denote as *m*-point-SPIR, and which can be used when the server's database contains N items, of which at most $m = o(N / \log N)$ are unique and the other $N - m$ items have some default value. The receiver does not know whether it learns a unique or a default value. We show a protocol which is based on HDOT and can be reduced to oblivious transfer alone, which computes this functionality more efficiently than known PIR protocols. *m*-point-SPIR can be used for other applications, as described in Section 6.

- A specific application that we describe in more detail checks whether two documents are similar. This application is novel in that it adds privacy to state-of-the-art document similarity algorithms of Broder et al. (1997). We designed and also implemented secure protocols for this task and ran experiments which demonstrate their efficiency. These protocols work in the semi-honest setting.

2 Preliminaries

We use the standard definitions of secure two-party computation in the stand-alone setting [see Goldreich's (2004) book (Chapter 7)]. Security of protocols is analysed by comparing what an adversary can do in a real execution of the protocol to what it can do in an ideal scenario that is secure by definition. The ideal scenario involves an incorruptible trusted third party (TTP) which receives the inputs of the parties, computes the desired functionality, and returns to each party its respective output. A protocol is secure if any adversary which participates in the real protocol (where no TTP exists) can do no more harm than if it was involved in the above-described ideal computation. The exact definition appears in Goldreich (2004).

The hybrid model. Our protocols use other secure protocols, such as oblivious transfer, as subprotocols. It has been shown in Canetti (2000) that if the subprotocols are secure according to the right definition (i.e., full simulatability in the case of the malicious adversary scenario), it suffices to analyse the security of the main protocol in a hybrid model. In this model, the parties interact with each other and have access to a trusted party that computes for them the functionalities that are implemented by the subprotocols. The composition theorem states that it is not required to analyse the execution in the real model, but rather only compare the execution in the hybrid model to that in the ideal model.

We remark that the composition theorem of Canetti (2000) holds for the case that the subprotocol executions are all run sequentially (and the messages of the protocol calling the subprotocol do not overlap with any execution). We also remark that if the oblivious transfer subprotocol is secure under parallel composition, then it is straightforward to extend (Canetti, 2000) so that the subprotocols may be run in parallel (again, as long as the messages of the protocol calling the subprotocol do not overlap with any execution).

2.1 Cryptographic primitives and tools

Homomorphic encryption. A homomorphic encryption scheme allows to perform certain algebraic operations on an encrypted plaintext by applying an efficient operation to the corresponding ciphertext. In addition, we require in this paper that the encryption scheme be *semantically secure*. In particular, we use an additively homomorphic encryption schemes where the message space is a ring (or a field). There, therefore, exists an algorithm $+_{pk}$ whose input is the public key of the encryption scheme and two ciphertexts, and whose output is $E_{pk}(m_1 + m_2) = E_{pk}(m_1) +_{pk} E_{pk}(m_2)$. (Namely, given the public key alone this algorithm computes the encryption of the sum of the plaintexts of two ciphertexts.) The new ciphertext is an encryption which is done with fresh and independent randomness. There is also an efficient algorithm $\cdot_{pk}$, whose input consists of the public key of the encryption scheme, a ciphertext, and a constant c in the field, and whose output is $E_{pk}(c \cdot m) = c \cdot_{pk} E_{pk}(m)$.

An efficient implementation of an additive homomorphic encryption scheme with semantic security was given by Paillier (1999, 2000). In this cryptosystem, the encryption of a plaintext from $[0, N-1]$, where N is an RSA modulus, requires two exponentiations modulo N^2. Decryption requires a single exponentiation. The Damgård-Jurik cryptosystem (Damgård and Jurik, 2001) is a generalisation of the Paillier cryptosystem that encrypts messages from the field $[1, N^s]$ using computations modulo N^{s+1}, where N is an RSA modulus and s a natural number. It enables more efficient encryption of larger plaintext. Security is based on the decisional composite residuosity (DCR) assumption.

Oblivious transfer. Oblivious transfer (abbrev. OT) refers to several types of two-party protocols where at the beginning of the protocol one party, a *sender*, has an input, and at the end of the protocol the other party, the *receiver*, learns some information about this input in a way that does not allow the sender to figure out what the receiver has learned. This paper uses 1-out-of-N oblivious transfer (OT_1^N) as a basic building block. The OT_1^N protocol runs between two parties, a sender that has an input $(X_0, X_1, \ldots, X_{N-1})$, where $X_i \in \{0, 1\}^m$, and a receiver that has an input $I \in \{0, 1, \ldots, N-1\}$. By the end of the protocol, the receiver learns X_I and nothing else and the sender does not learn any information about I. In Naor and Pinkas (2005), it was shown how to implement OT_1^N using $\log N$ invocations of OT_1^2. (In a nutshell, this transformation works by using $\log N$ pairs of keys, where each combination of $\log N$ keys encrypts a different input, and then letting the receiver learn a single key out of each pair.) There are many efficient implementations of OT_1^2, starting with a protocol of Even et al. (1982). Most of these protocols are designed for the semi-honest scenario, or for a malicious scenario where the protocol provides only the privacy property and not full simulatability. We note that while our protocol for the semi-honest scenario can use any OT protocol, the protocol for the malicious adversary scenario must use an OT protocol which is secure in the sense of full simulatability against malicious adversaries. Such protocols were described (e.g., in Camenisch et al., 2007; Green and Hohenberger, 2007; Peikert et al., 2008; Hazay and Lindell, 2008). (We specifically need a committed OT variant where we can also prove a relation between the inputs of the sender, and therefore, we use a protocol which builds on the work of Jarecki and Shmatikov (2007) We also note that in the malicious case we use OT_1^2 and not OT_1^N.

The implementation that we ran (and which works in the semi-honest scenario) uses the protocol of Bellare and Micali (1990), but can be based on any OT protocol (e.g., those of Naor and Pinkas, 2001; Aiello et al., 2001).

Instead of using OT_1^n, one could use a symmetric PIR protocol, SPIR, which has $o(n)$ communication overhead and also guarantees that the server learns only a single item of the sender's inputs (Gertner et al., 1998). Any PIR scheme can be translated to a symmetric PIR scheme (Naor and Pinkas, 2005). Therefore, PIR protocols with sublinear or polylogarithmic communication overhead (Kushilevitz and Ostrovsky, 1997; Cachin et al., 1999; Lipmaa, 2005; Gentry and Ramzan, 2005) yield symmetric PIR protocols with the same overhead. Unfortunately, these PIR protocols require executing $O(n)$ exponentiations (compared to $O(n)$ symmetric encryption operations in OT_1^n protocols). We think that this computation overhead might be prohibitive for implementations, and therefore only describe and analyse the usage of OT.

Preprocessing. The operation of any protocol can be potentially improved by moving part of the computation to a preprocessing phase, i.e., a step that is run before the parties receive their inputs. (Another name for preprocessing is offline/online computation, where preprocessing can be performed offline, and online operation happens after the input is received.) Running a preprocessing step lets the parties perform part of the computation in a time which is most convenient for them, and reduces the overhead incurred after receiving the inputs.

It is important to distinguish between *interactive* and *non-interactive* preprocessing. The former requires the parties to communicate with each other before receiving their inputs, while the latter lets each party do its preprocessing by itself. It is of course preferable to use non-interactive preprocessing, and we demonstrate how it can be applied to the protocols that we present [this is preferable to methods that improve the overhead of OT by performing *interactive* preprocessing, e.g., using the 'extended OT' protocols of Beaver (1996) and Ishai et al. (2003)].

2.2 Related work

Generic secure computation. Generic protocols (e.g., of Yao 1986) can be used to compute any function. They are typically based on representing the computed function as a binary or an algebraic circuit, and applying the protocol to this representation. The overhead of these protocols depends on the size of the circuit representation of the functions. There are many theoretical constructions of secure generic protocols. Notable examples of *implementations* of secure computation are the Fairplay system (Malkhi et al., 2004) for secure two-party computation, and the FairplayMP and SIMAP systems (Ben-David et al., 2008; Bogetoft et al., 2006) for secure multi-party computation. The system described in Lindell et al. (2008) and Pinkas et al. (2009) implements fully simulatable secure two-party computation according to the recent construction of Lindell and Pinkas (2007). For certain specific functions, there are specialised protocols which are more efficient than the generic constructions. Such functions include for example, equality testing (Fagin et al., 1996), or set intersection (Freedman et al., 2004).

Computing the Hamming distance and computing similarity. Protocols for computing the scalar product of vectors (which is equal to the Hamming distance if the alphabet is binary) were suggested in Wright and Yang (2004), and Goethals et al. (2004). These protocols are based on the use of homomorphic encryption, and are only secure against semi-honest adversaries. (Our HDOT protocol for the case of binary alphabets and semi-honest adversaries borrows its first step from these protocols.)

A protocol for secure efficient *approximate* computation of the Hamming distance, with a polylogarithmic communication overhead, was suggested in Indyk and Woodruff (2006) [previous protocols for this task use $O(\sqrt{\ell})$ communication for ℓ-bit words (Feigenbaum et al., 2006; Freedman et al., 2004)]. We wanted to improve upon these protocols for three reasons:

1. These protocols introduce approximation errors.
2. The protocols are only secure against semi-honest adversaries.
3. In addition, these protocols have good asymptotic communication overhead, but use non-trivial components which seem difficult to implement with a performance that will be competitive for reasonable input sizes.[2] (We note that another difficulty in using these protocols is that they output an approximation of the Hamming distance itself, rather than outputting a function of the approximated distance. It seems possible, however, to adapt the protocols to the latter requirement.)

For the application of secure computation of similarity, it was shown by Charikar (2002) that the Jaccard similarity distance embeds isometrically into ℓ_1. This means that similarity can be computed as the Hamming distance between two binary vectors. Our paper uses a more straightforward method which applies multiple permutations to the compared inputs, and then computes the Hamming distance between two vectors over a large alphabet.

3 Hamming distance-based oblivious transfer

A HDOT protocol is run between two parties, a receiver $(\mathcal{P}_1)$ and a sender $(\mathcal{P}_2)$. It is defined as follows:

- Input: $\mathcal{P}_1$'s input is a word $w \in \Sigma^{\ell}$. $\mathcal{P}_2$'s input contains a word $w' \in \Sigma^{\ell}$, and $\ell + 1$ values $Z_0, \dots Z_{\ell}$.
- Output: $\mathcal{P}_1$'s output is Z_d, where $d = d_H(w, w')$ is the Hamming distance between w and w' (note that $\mathcal{P}_1$ does not learn the Hamming distance itself). $\mathcal{P}_2$ has no output.

In other words, the HDOT functionality can be described as the following mapping

$$\left(w, \left[w', \left(Z_0, \dots, Z_{\ell}\right)\right]\right) \mapsto \left(Z_{d_H(w,w')}, \perp\right)$$

This paper describes a special protocol, *bin*HDOT, for the case that the input words are binary (i.e., $\Sigma = \{0, 1\}$), and a general protocol which works for alphabets Σ of arbitrary size.

3.1 Straightforward applications

An HDOT protocol can be immediately used for computing any function which depends on the Hamming distance. Following are some interesting examples of such functions:

- The *Hamming distance* itself can be computed by setting $Z_i = i$ for every $0 \le i \le \ell$.
- The *parity* of the exclusive-or of the two inputs is computed by setting Z_i to be equal to the least significant bit of i, for $0 \le i \le \ell$.
- *EQ – equality-based transfer*, or $EQ_{\mathcal{V}_0,\mathcal{V}_1}(w, w')$: This functionality outputs $\mathcal{V}_0$ if $w = w'$, and $\mathcal{V}_1$ otherwise. The functionality is computed by setting $Z_0 = \mathcal{V}_0$ and $Z_i = \mathcal{V}_1$ for $1 \le i \le \ell$, and executing an HDOT protocol. $\mathcal{P}_1$ does not know which of the two cases happens (namely, whether $w = w'$). This is crucial for the applications that are described below.

  Recall that it is easy to design a protocol in which $\mathcal{P}_1$ learns a specific value $\mathcal{V}_0$ if the two inputs are equal, and a *random* value otherwise. [See Fagin et al. (1996), or consider a protocol where $\mathcal{P}_1$ sends a homomorphic encryption $E(w)$, and receives back $E(r \cdot (w - w') + \mathcal{V}_0)$, where r is a random value.] Our protocol is unique in defining a specific value to be learned if the two inputs are different, and in hiding whether the inputs are equal or not.[3]

- *Threshold HDOT protocol*: The equality-based transfer protocol can be generalised to tolerate some errors and have the output be $\mathcal{Y}_0$ if the Hamming distance is smaller than a threshold τ, and be $\mathcal{Y}_1$ otherwise. In other words, it implements the following functionality:

$$HDOT^{\tau}_{\mathcal{Y}_0|\mathcal{Y}_1}\left(w, w'\right) = \begin{cases} \mathcal{Y}_0, & d_H\left(w, w'\right) < \tau \\ \mathcal{Y}_1, & d_H\left(w, w'\right) \geq \tau \end{cases}$$

  This functionality is implemented by setting $Z_0 = \cdots = Z_{\tau-1} = \mathcal{Y}_0$, and $Z_\tau = \cdots = Z_\ell = \mathcal{Y}_1$.

The protocol for equality-based transfer is the major building blocks of the m-point-SPIR application described in Section 6.

4 Protocols secure against semi-honest adversaries

We first describe protocols which are secure against semi-honest behaviour of the potential adversaries. These protocols are relatively simple yet they are unique in invoking oblivious transfer a number of times which is only logarithmic in the input length. The malicious adversary scenario is covered in Section 5.

4.1 A protocol for binary alphabets (binHDOT)

Consider first the case where the alphabet is binary ($\Sigma = \{0, 1\}$). The *bin*HDOT functionality can be securely implemented by applying Yao's protocol to a circuit computing it. That solution would require running ℓ invocations of OT_1^2. We describe here a protocol which accomplishes this task using only $\log(\ell + 1)$ OT_1^2s (see below a comparison of the performance of these two protocols).

The protocol works in the following way: In the first step, the parties use homomorphic encryption to count the number of bits in which the two words differ. The result is in the range $[0, \ell]$. Next, the two parties use $\mathrm{OT}_1^{\ell+1}$ (implemented using $\log(\ell + 1)$ OT_1^2s) to map the result to the appropriate output value. The protocol is described in detail in Figure 1.[4]

Correctness: The value d_H is equal to the Hamming distance. In Step 4, $\mathcal{P}_1$ computes (in $\mathcal{F}$) the value $d_H + r$, which can have one of $\ell + 1$ values (namely $r, r + 1, \ldots, r + \ell$). It holds with probability $1-\ell/|\mathcal{F}|$ that $r < |\mathcal{F}|-\ell$. (And since $|\mathcal{F}|$ is typically very large compared to ℓ, e.g., $|\mathcal{F}| \approx 2^{1024}$ and $\ell < 1{,}000$, we do not consider here the negligible probability that this event does not happen.) Therefore, the computation of $d_H + r$ in $\mathcal{F}$ does not involve a modular reduction and has the same result as adding them over the integers. Reducing the result modulo $(\ell + 1)$ (in Step 5) is therefore equal to $(r + d_H) \bmod (\ell + 1)$. $\mathcal{P}_1$ uses this result as its input to the 1-out-of-$(\ell + 1)$ OT protocol of Step 5. $\mathcal{P}_2$, on the other hand, sets the sender's inputs in the OT such that each Z_i value is the sender's input indexed by $(r + i) \bmod (\ell + 1)$. As a result, the output of $\mathcal{P}_1$ in the OT protocol is Z_{d_H}, as required.

Note that if the parties are only interested in computing the value of the Hamming distance then the protocol can be greatly simplified: $\mathcal{P}_2$ should send to $\mathcal{P}_1$ in Step 3 the encryption $E_{pk}(d_H)$. There is no need to run Steps 4 and 5.

Improving the initial step using non-interactive preprocessing. An additional improvement can be achieved in the first step of the protocol, where $\mathcal{P}_1$ sends an encrypted binary representation of the word. This representation can be precomputed using *non-interactive* preprocessing: $\mathcal{P}_1$ can prepare in advance ℓ encrypted zeros and ℓ encrypted ones, instead of encrypting the input bits online. This preprocessing enables $\mathcal{P}_1$ to send the binary representation directly without spending time online encrypting 0 and 1 values.

Overhead. We compare the overhead of the *bin*HDOT protocol to that of applying Yao's protocol to a circuit computing the same functionality. We note that the runtime of an OT protocol is slower than that of a homomorphic encryption or decryption, and that the runtime of these latter operations is *much* slower than that of a homomorphic addition or a homomorphic multiplication by a constant (which in turn is *much* slower than symmetric encryption or decryption). This relation between run times can be summarised as follows (where > denotes 'slower', and >> denotes slower by an order of magnitude):

OT > homomorphic enc. >> homomorphic addition
>> symmetric enc.

Without using any preprocessing, the binHDOT protocol requires $\mathcal{P}_1$ to compute ℓ encryptions and a single decryption, while $\mathcal{P}_2$ computes $\ell + 1$ homomorphic additions, and the two parties run $\log(\ell + 1)$ $\mathrm{OT}_1^2 s$ and $(\ell + 1)$ symmetric encryptions (in order to implement $\mathrm{OT}_1^{\ell+1}$). In Yao's protocol, the parties compute a circuit with ℓ input bits and a total of $O(\ell)$ gates. This requires ℓ executions of an OT_1^2 protocol and $O(\ell)$ symmetric encryptions and decryptions. Both protocols require $O(\ell)$ communication.

The improvement achieved by the *bin*HDOT protocol is noticeable since it reduces the number of OTs, which are the most time consuming operation, from ℓ to $\log(\ell + 1)$. In addition, the *bin*HDOT protocol can benefit from the use of *non-interactive* preprocessing to precompute all homomorphic encryption operations even before the parties know of each other. In that case the ℓ encryptions done by $\mathcal{P}_1$ are computed offline, and its online computation is composed of a single decryption and $\log(\ell + 1)$ OTs. (Yao's protocol cannot precompute the oblivious transfers without using interaction. We note that if interactive preprocessing is possible, then the OTs themselves can be precomputed in both protocols, and this reduces the overhead of both protocols.)

Theorem 1: The *bin*HDOT protocol in Figure 1 is secure against semi-honest adversaries in the OT hybrid model (i.e., under the assumption that the OT protocol is secure).

Proof: The security analysis is done under the assumption that the parties are semi-honest and that the OT protocol is implemented by an oracle (the latter assumption can be replaced by using an OT protocol secure against semi-honest adversaries). In the protocol, $\mathcal{P}_2$ receives homomorphic encryptions of a binary representation of a word, and then it plays the role of the sender in an OT protocol in which it receives no output. We can simulate $\mathcal{P}_2$'s view by sending it encryptions of random values. If $\mathcal{P}_2$ can distinguish between these encryptions and the encryptions it receives in the protocol, then a standard reduction shows, through a hybrid argument, that $\mathcal{P}_2$ can break the semantic security of the encryption. As for $\mathcal{P}_1$, it receives from $\mathcal{P}_2$ a random value ($d_H + r$) and then it participates as the receive in the OT protocol. The parties are semi-honest and therefore they follow he directions of the protocol and thus the output of the OT is the designated output of the protocol. It is therefore possible to simulate $\mathcal{P}_1$'s view by sending it first a random value, and then send it, as the output of the OT, the output of the functionality (learned in the simulation from the TTP). □

Figure 1 The *bin*HDOT protocol secure against semi-honest adversaries

*bin*HDOT$_{\langle Z_0,\ldots,Z_\ell\rangle}$ (w, w') Protocol

Input: $\mathcal{P}_1$'s input is a word $w = (w_0, \ldots, w_{\ell-1})$, $\mathcal{P}_2$'s input is $w' = (w'_0, \ldots, w'_{\ell-1})$, where w_i; $w'_i \in \{0, 1\}$. $\mathcal{P}_2$ has additional inputs $(Z_0, \ldots, Z_\ell)$.

Output: $\mathcal{P}_1$ receives Z_i such that $d_H(w, w') = i$. $\mathcal{P}_2$ learns nothing.

The protocol uses $E_{pk}(\cdot)$, a homomorphic encryption function. The plaintexts are in a ring or a field $\mathcal{F}$. (We emphasise that ℓ and $|\Sigma|$ are negligible compared to $|\mathcal{F}|$. A typical size could be $|\mathcal{F}| = 2^{1024}$.) *pk* is a public key that both parties know, but only $\mathcal{P}_1$ knows the corresponding private key and can decrypt messages.

1 $\mathcal{P}_1$ sends the homomorphic encryption of each bit of the binary representation of $w = \{w_0, \ldots, w_{\ell-1}\}$, where $w_i \in \{0, 1\}$.

2 $\mathcal{P}_2$ receives the encrypted representation $\{E_{pk}(w_0), \ldots, E_{pk}(w_{\ell-1})\}$. For each bit location j it calculates $E_{pk}(\vartheta_j)$, where $\vartheta_j \in \{0, 1\}$ and is equal to 1 if, and only if, $w_j \neq w'_j$. The calculation is done in the following way:

$$\begin{aligned} E_{pk}(\vartheta_j) &= E_{pk}\left(w_j \cdot (1 - w'_j) + 1(1 - w_j) \cdot w'_j\right) \\ &= E_{pk}(w_j) \cdot_{pk} (1 - w'_j) +_{pk} \left(1 -_{pk} E_{pk}(w_j)\right) \cdot_{pk} w'_j \end{aligned}$$

At the end of this step, $\mathcal{P}_2$ has the values $\{E_{pk}(\vartheta_0), \ldots, E_{pk}(\vartheta_{\ell-1})\}$.

3 Using the homomorphic properties, $\mathcal{P}_2$ sums the results of the previous step and computes $E_{pk}(d_H) = \sum_0^{\ell-1} E_{pk}(\vartheta_i)$. The value d_H is in the range $\{0, 1, \ldots, \ell\}$ and is equal to the Hamming distance between the two input words. In addition, $\mathcal{P}_2$ chooses a random value $r \in \mathcal{F}$, computes the value $E_{pk}(d_H + r)$, and sends it to $\mathcal{P}_1$. (In other words, it shifts the result by a random value r. Note that with overwhelming probability, $1 - \ell / |\mathcal{F}|$, this addition operation does not involve a modular reduction.)

4 $\mathcal{P}_1$ receives $E_{pk}(d_H + r)$ and decrypts the result.

5 Next, the parties map the result to the appropriate Z_i value, by invoking a $OT_1^{\ell+1}$ protocol where $\mathcal{P}_1$ is the receiver and $\mathcal{P}_2$ is the sender:

- The input of $\mathcal{P}_1$ is $(d_H + r) \bmod (\ell + 1)$.
- $\mathcal{P}_2$ has inputs $X_0, \ldots, X_\ell$, where $X_i = Z_{(i-r) \bmod (\ell+1)}$ (namely, Z_i is mapped to input $(i + r) \bmod (\ell + 1)$ of the OT).

$\mathcal{P}_1$'s output in the OT is its output in the *bin*HDOT protocol.

Figure 2 The HDOT protocol for general alphabets

HDOT$_{\langle Z_0,\ldots,Z_\ell\rangle}$ (w, w') Protocol

Input: $\mathcal{P}_1$ has an input $w = \langle w_0, w_1, \ldots, w_{\ell-1}\rangle \in \Sigma^\ell$. $\mathcal{P}_2$ has an input $w' = \langle w'_0, w'_1, \ldots, w'_{\ell-1}\rangle \in \Sigma^\ell$, and additional input values $Z_0, \ldots, Z_\ell$. We denote by $\bar{w}_j$ the binary representation of w_j, which is $\lceil \log(|\Sigma|)\rceil$ bits long.

Output: $\mathcal{P}_1$ learns Z_i such that $d_H(w, w') = i$, $\mathcal{P}_2$ learns nothing.

1 For every $i \in [0, \ell - 1]$, $\mathcal{P}_2$ chooses at random a value $\alpha_i \in_R \mathcal{F}$. Both parties then run the protocol $EQ_{\alpha_i,\alpha_{i+1}}(\bar{w}_i, \bar{w}'_i)$. ($\bar{w}_i, \bar{w}'_i$ denote the binary representations of the letters w_i and w'_i, respectively. The output of this protocol is α_i if $w_i = w'_i$, and $\alpha_i + 1$ otherwise.)

At the end of the process, $\mathcal{P}_1$ obtains the values $\{\beta_0, \ldots, \beta_{\ell-1}\}$, where

$$\beta_i = \begin{cases} \alpha_i, & w_i = w'_i \\ \alpha_i + 1, & w_i \neq w'_i \end{cases}$$

2 $\mathcal{P}_1$ sums, modulo $(\ell + 1)$, the β_i values it received. Namely, it computes $\sigma_\beta = \left(\sum_0^{\ell-1} \beta_i\right) \bmod (\ell + 1)$. $\mathcal{P}_2$ sums its α values and computes $\sigma_\alpha = \left(\sum_0^{\ell-1} \alpha_i\right) \bmod (\ell + 1)$.

3 Both parties run an $OT_1^{\ell+1}$ protocol with the following inputs:

- $\mathcal{P}_1$ is the receiver and its input is σ_β
- $\mathcal{P}_2$ is the sender and its input is $\{X_0, \ldots, \ldots, X_\ell\}$, where $X_i = Z_{(i-\sigma_\alpha) \bmod (\ell+1)}$.

The value that $\mathcal{P}_1$ receives in the OT is defined as its output in the protocol.

4.2 A protocol for arbitrary alphabets (HDOT)

We now describe an HDOT protocol which works over arbitrary alphabets Σ. The protocol is based on applying the *bin*HDOT protocol to every character of the words. More specifically, the parties have inputs $w, w' \in \Sigma^\ell$, respectively. The protocol begins with the parties representing each of the letters of Σ as a binary word of length $\lceil \log |\Sigma| \rceil$, and then running (for each letter location) the equality-based transfer (*EQ*) protocol, which was defined above and is an application of *bin*HDOT. In each execution of the *EQ* protocol $\mathcal{P}_1$ learns a value α_i if $w_i = w'_i$, or the value $\alpha_i + 1$ otherwise, where α_i is chosen at random by $\mathcal{P}_2$. Then, $\mathcal{P}_1$ sums the values that it has received modulo $\ell + 1$. The result is equal, modulo $\ell + 1$, to $\sum \alpha_i$ plus the Hamming distance of the original words. The parties then run an OT$_1^{\ell+1}$ protocol to map the result to the desired output. The protocol is detailed in Figure 2.

Correctness. For every $0 \leq i \leq \ell - 1$, $\mathcal{P}_1$ and $\mathcal{P}_2$ learn in Step 1 values β_i, α_i, respectively, such that $\beta_i = \alpha_i$ if the letters w_i and w'_i are equal, and $\beta_i = \alpha_i + 1$ if the letters are different. Let $S_\alpha = \sum_{i=0}^{\ell-1} \alpha_i$, where here the addition is done in $\mathcal{F}$. Define S_β similarly. Let d be the Hamming distance between the two input words. Then it holds with probability $1 - \ell / |\mathcal{F}|$ that $S_\beta = S_\alpha + d$, where the addition here is done over the integers. Therefore, the values $\sigma_\alpha = S_\alpha \bmod (\ell + 1)$ and $\sigma_\beta = S_\alpha \bmod (\ell + 1)$ computed in Step 2 satisfy that $\sigma_\beta - \sigma_\alpha \bmod (\ell + 1)$ is equal to the Hamming distance d (which is in the range $[0, \ell]$).

Consider now the OT in Step 3. Assume first that $\sigma_\alpha = 0$. In this case $\mathcal{P}_1$'s input to the OT, σ_β, is equal to the Hamming distance, and the inputs of $\mathcal{P}_2$ to the OT are the values $Z_0, \ldots, Z_\ell$ (in that order). The OT protocol therefore computes the desired output in this case. Now, if $\sigma_\alpha > 0$ then $\mathcal{P}_1$'s input to the OT protocol is cyclically shifted (modulo $\ell + 1$) by σ_α, while the order of $\mathcal{P}_2$'s inputs to the OT is also cyclically shifted (modulo $\ell + 1$) by the same value σ_α. The OT protocol therefore computes the correct result.

Overhead. The overhead is that of applying the *bin*HDOT protocol ℓ times over log $|\Sigma|$ long binary strings, and then running $\log(\ell + 1)$ invocations of OT$_1^2$. The parties run $\ell \log \log |\Sigma| + \log(\ell + 1)$ OT$_1^2$s, as well as $O(\ell \log |\Sigma|)$ homomorphic operations. (A direct implementation of this functionality using Yao's protocol would have required invoking $O(\ell \log |\Sigma|)$ OTs.)

Theorem 2: The HDOT protocol in Figure 2 is secure against semi-honest adversaries in the OT hybrid model.

Proof: Analysing security in the hybrid model, we assume that the OT and *bin*HDOT protocols, and therefore also the *EQ* protocol, are executed by a trusted oracle. $\mathcal{P}_2$ is the sender in the *bin*HDOT and OT protocols. Therefore, it does not learn any information by participating in these protocols. $\mathcal{P}_1$ receives in Step 1 the β_i values, which are defined as either $\beta_i = \alpha_i$ or $\beta_i = \alpha_i + 1$, where each α_i value is chosen randomly by $\mathcal{P}_2$. In the last step, $\mathcal{P}_1$ receives in the OT the result of mapping the sum of the values to the appropriate Z_i value, which is the designated output of the protocol. We can therefore simulate $\mathcal{P}_1$'s view by first sending it random values and then sending it the output of the functionality (as learned from the TTP used in the simulation). □

4.3 Weighted Hamming distance-based OT

The *weighted* Hamming distance between two ℓ-letter strings w, w' is defined in the following way: The function depends on a set of integer weights $\omega_0, \ldots, \omega_{\ell-1}$. We define δ_i, for $0 \le i \le \ell - 1$, to be 0 if $w_i = w_i'$, and 1 otherwise. The weighted Hamming distance is $\sum_{i=0}^{\ell-1} \delta_i \omega_i$ (earlier we handled the case where $\forall i\ \omega_i = 1$). This function enables to assign to any letter location a specific weight corresponding to its importance.

It is possible to slightly change the HDOT protocols to support the computation of a weighted Hamming distance-based OT. In the binary alphabet case, the revised *bin*HDOT protocol computes in Step 2 the values $E_{pk}(\vartheta_j \omega_j)$ by multiplying $E_{pk}(\vartheta_j)$ by ω_i. The value d_H is defined to be the sum of these values. Let $\Omega = \sum_{i=0}^{\ell-1} \omega_i$. The value of d_H is in the range [0, Ω]. Therefore, $\mathcal{P}_2$ has inputs $Z_0, \ldots., Z_\Omega$, and the last step of the protocol computes a 1-out-of-(Ω + 1) OT. In the case of an arbitrary alphabet, each β_i value is set to $\alpha_i + \omega_i$ if the two letters are different, and to α_i is they are equal. Again, the last step computes a 1-out-of-(Ω + 1) OT.

5 A *bin*HDOT protocol for malicious adversaries

Designing an efficient protocol which is secure against malicious adversaries [in the sense of full simulatability, as defined in Goldreich (2004)] is a challenging task. A protocol with this level of security can be implemented using generic constructions, such as the constructions in Lindell and Pinkas (2007), and Jarecki and Shmatikov (2007), but these currently impose an additional overhead, caused, for example, by communicating and evaluating multiple copies of a circuit computing the functionality. We design a new *bin*HDOT protocol to handle the presence of malicious adversaries. In this protocol, the parties use committed OT to learn whether corresponding bits of the two words are equal, and then use an OPE protocol (Naor and Pinkas, 1999; Hazay and Lindell, 2008) to map the result to an output value. (This is different than the semi-honest case, where homomorphic encryption was used to compare bits, and OT_1^N was used to compute the final result.) The new protocol uses OT and OPE protocols which are efficient and yet are secure in the sense of full simulatability against malicious adversaries. Security can therefore be analysed in the hybrid model. In more detail, the protocol uses the following tools:

Committed 1-out-of-2 oblivious transfer with constant difference. (or COTCD_1^2), secure against malicious adversaries. A committed OT protocol in an OT protocol where the parties commit to their inputs: the sender commits to its inputs m_0, m_1 and the receiver commits to its input $\sigma \in \{0, 1\}$. During the protocol, each party can verify that the other party's input is equal to the corresponding committed value. We define a committed OT with constant difference (COTCD, pronounced 'cot-cd') to be a committed OT with an additional auxiliary input composed of a value Δ known to the sender, and a commitment to Δ which is known to the receiver. The protocol lets the receiver verify that the difference of the two inputs of the sender is ±Δ. In other words, it either holds that $m_1 - m_0 = \Delta$ or that $m_0 - m_1 = \Delta$. (In our application the COTCD protocol will be run several times with the same committed Δ value. The protocol will be run once for each letter location. The receiver learns some value if the two corresponding letters are equal. If the letters are different it learns that value plus Δ. Summing the values learned for all letters, the receiver obtains a result which is equal to some base value plus Δ times the Hamming distance. This value is then used for computing the final result.)

We describe in Appendix how to construct the COTCD primitive based on the Jarecki and Shmatikov (JS) (2007) committed OT protocol, which is in turn based on the Camenisch-Shoup (CS) encryption scheme (Camenisch and Shoup, 2003).[5] We use that protocol since it can be used to transfer strings, and since it is easy to add to it an efficient zero-knowledge proof that the messages of the sender have the required difference [it seems much harder to add a proof of this type to other OT protocols which are secure against malicious adversaries, such as the protocols of Hazay and Lindell (2008), Peikert et al. (2008)]. The JS protocol is UC-secure in the common reference string model and therefore all invocations of that protocol can be run in parallel. As a result, the HDOT protocol we construct can execute in parallel all ℓ invocations of the COTCD protocol. Alternatively, the two parties can run a secure protocol for computing the common random string (CRS) required for UC-security, and obtain a protocol which is secure in the standard model (see Appendix). The protocol is proved to be secure under the DCR assumption (i.e., the assumption on which the Paillier homomorphic encryption system is based).

Constrained OT: COTCD is an example on a family of oblivious transfer protocols which we can denote as 'constrained OT'. This family contains OT protocols which have additional constraints on the values of their inputs and where the receiver verifies that these constraints hold. In the case of COTCD, the two input values of the sender must have a difference which is equal to the committed value Δ. (Another example is the circuit evaluation protocol of Jarecki and Shmatikov (2007), where the constraint is that the values transferred in the OT can decrypt entries in gate tables.)

Commitment scheme. The CS encryption scheme (Camenisch and Shoup, 2003) is used in our protocol as a commitment scheme, as is suggested in Jarecki and Shmatikov (2007). The details are described in Appendix.

An OPE protocol. (Secure against malicious adversaries). An OPE protocol (Naor and Pinkas, 1999) is a protocol where the sender's input is a polynomial $P(\cdot)$ of a certain degree, and the receiver's input is a value x. The receiver's output is $P(x)$ while the sender learns nothing. We use the OPE construction of Hazay and Lindell (2008), which is secure (in the sense of full simulatability) against malicious adversaries, and uses very few exponentiations.

The underlying fields. The output of the COTCD protocol is used as an input of the OPE protocol. The COTCD protocol runs in a group $\mathcal{F} = \mathbb{Z}^*_{n^2}$, where $\mathbb{Z}^*_{n^2}$ is defined by a safe RSA modulus $n = pq$, where $p = 2p' + 1$, $q = 2q' + 1$, $|p| = |q|$, $p \neq q$ and p, q, p', q' are all primes. The encryption scheme of Camenisch and Shoup, which is used in the protocol as a commitment scheme, works in the same group. The OPE protocol of Hazay and Lindell (2008) runs in $\mathbb{Z}_N$, with N being an RSA modulus. Our protocol must enable the parties to use the result of the COTCD protocol as an input to the OPE protocol. It must therefore use a group $\mathbb{Z}^*_{n^2}$ and a field $\mathbb{Z}_N$, which satisfy that $|\mathbb{Z}^*_{n^2}| < |\mathbb{Z}_N|$, and therefore we will require that $n^2 < N$. We define a simple mapping $f : \mathbb{Z}^*_{n^2} \to \mathbb{Z}_N$, where the only requirement is that no two elements of $\mathbb{Z}^*_{n^2}$ are mapped by f to the same value in $\mathbb{Z}_N$. The protocol then performs the initial computations in $\mathbb{Z}^*_{n^2}$ and then uses f to map the result to $\mathbb{Z}_N$.

The protocol itself is described in Figure 3. In the protocol, for every bit location i, $\mathcal{P}_1$ receives a value t_i^0 if the corresponding bits are equal, and the value $t_i^0 + \Delta$ otherwise. The value Δ, and also all t_i^0 values, are randomly chosen by $\mathcal{P}_2$. (In the semi-honest case $\mathcal{P}_1$ learned one of two values whose difference was 1. Here, the difference is a random number Δ in order to prevent attacks by a malicious $\mathcal{P}_1$.) $\mathcal{P}_1$ then sums the values it received, and obtains the result $\sum_{i=1}^{\ell} t_i^0 + d \cdot \Delta$, where d is the Hamming distance. We use the notation $\sigma_r = \sum_{i=1}^{\ell} t_i^0$. $\mathcal{P}_2$ then prepares an OPE where $\forall j \in [0, \ell]$, $P(f(\sigma_r + j \cdot \Delta)) = Z_j$. The parties execute an OPE and $\mathcal{P}_1$ computes $P(f(\sigma_r + d\Delta))$ and learns the desired result.

Figure 3 The *bin*HDOT protocol for the malicious case

Malicious *bin*HDOT$_{\langle Z_0,\ldots,Z_\ell \rangle}$ (w, w') Protocol

Input: $\mathcal{P}_1$'s input is a word $w = (w_0, \ldots, w_{\ell-1})$, $\mathcal{P}_2$'s input is $w' = (w'_0, \ldots, w'_{\ell-1})$, where $w_i,\ w'_i \in \{0, 1\}$. $\mathcal{P}_2$ has additional inputs $(Z_0, \ldots, Z_\ell)$.

Output: $\mathcal{P}_1$ receives Z_i such that $d_H(w, w') = i$ (i.e., the Hamming distance of w and w' is i). $\mathcal{P}_2$ learns nothing.

1 $\mathcal{P}_2$ chooses at random $\Delta \in_R \mathbb{Z}^*_{n^2}$ and sends to $\mathcal{P}_1$ a commitment to Δ. In addition it proves to $\mathcal{P}_1$, using a zero-knowledge proof of knowledge, the knowledge of Δ.

2 For each pair of bits $(w_i,\ w'_i)$, both parties use COTCD to check whether the bits are equal:

- $\mathcal{P}_2$ chooses a random value $t_i^0 \in_R \mathcal{F}$, and defines $t_i^1 = t_i^0 + \Delta$.
- Both parties run a COTCD protocol:
  - (a) The auxiliary inputs to the protocol are Δ, known to $\mathcal{P}_2$, and a commitment to Δ, known to $\mathcal{P}_1$.
  - (b) $\mathcal{P}_1$ is the receiver and its input is w_i.
  - (c) $\mathcal{P}_2$ is the sender. If $w'_i = 0$ then it sets $(x_i^0,\ x_i^1) = (t_i^0,\ t_i^1)$; Otherwise, $(x_i^0,\ x_i^1) = (t_i^0,\ t_i^1)$.

In each execution of the protocol, if both bits are equal then $\mathcal{P}_1$ learns t_i^0, otherwise, $\mathcal{P}_1$ learns t_i^1. (In addition, the verification step of the COTCD protocol enables $\mathcal{P}_1$ to verify that $|x_i^1 - x_i^0| = \Delta$. If this check fails then $\mathcal{P}_1$ aborts the protocol.)

By the end of this step, $\mathcal{P}_1$ learns $t_0^{b_0}, \ldots, t_{\ell-1}^{b_{\ell-1}}$, where $b_i = w_i \oplus w'_i$, while $\mathcal{P}_2$ does not learn any information.

3 $\mathcal{P}_1$ computes $\sigma_t = \sum t_i^{b_i}$ and $\mathcal{P}_2$ computes $\sigma_r = \sum t_i^0$. These summations are done in $\mathbb{Z}^*_{n^2}$.

4 $\mathcal{P}_2$ constructs a polynomial $P(x) = \sum_0^{\ell} a_i x^i$ in $\mathbb{Z}_N$, such that $P(f(\sigma_r + i \cdot \Delta)) = Z_i$, $\forall i \in \{0, 1, \ldots, \ell\}$ (where f is the simple mapping from $\mathbb{Z}^*_{n^2}$ to $\mathbb{Z}_N$), and $P(0)$ is random. (This construction succeeds if $0 \notin \{\sigma_r, \ldots, \sigma_r + \ell\Delta\}$, which happens with probability $1 - (\ell + 1)/|\mathbb{Z}_N|$.) The degree of P is $\ell + 1$.

5 $\mathcal{P}_1$ and $\mathcal{P}_2$ run an OPE protocol to evaluate $P(f(\sigma_t))$, such that $\mathcal{P}_1$ learns the result while $\mathcal{P}_2$ does not learn any information. (f is a mapping function as defined in Section 5)

The protocol uses an OPE instead of $\mathrm{OT}_1^{\ell+1}$ since the values are mapped to locations in a large range, rather than to indices in the range [0, ℓ], in order to prevent a malicious $\mathcal{P}_1$ from learning any Z_i value which does not correspond to the actual Hamming distance. If $\mathcal{P}_1$ evaluates the polynomial at any point other than intended, it is likely to receive a random answer since it does not know Δ and is therefore unlikely to choose any point corresponding to a Z_i value. As for a malicious $\mathcal{P}_2$, its inputs w' and $Z_0, \ldots, Z_\ell$ can be extracted from its interaction with the OT and OPE protocols, and are used for a simulation-based proof.

Theorem 3: (Correctness) The protocol of Figure 3 computes the *bin*HDOT functionality.

Proof: Let us follow the steps of the protocol. In each execution of the COTCD protocol, $\mathcal{P}_1$ learns t_i^0 if both bits are equal, otherwise, it learns $t_i^1 = t_i^0 + \Delta$. In other words, it learns $t_i^{b_i}$, where $b_i = w_i \oplus w_i'$.

Then, in Step 3, $\mathcal{P}_1$ computes $\sigma_t = t_0^{b_0} + \cdots + t_{\ell-1}^{b_{\ell-1}}$, and $\mathcal{P}_2$ computes $\sigma_r = t_0^0 + \cdots + t_{\ell-1}^0$. Therefore, it holds that $\sigma_t - \sigma_r = \Delta \cdot d_H(w, w')$. In Step 4, $\mathcal{P}_2$ constructs a polynomial $P(x)$ such that: $P(f(\sigma_r)) = Z_0$; $P(f(\sigma_r + \Delta)) = Z_1, \ldots, P(f(\sigma_r + \ell \cdot \Delta)) = Z_\ell$. In the last step of the protocol, the parties use an OPE protocol to compute $P(f(\sigma_t)) = Z_{d_H(w,w')}$. □

Theorem 4: (Security) The protocol securely computes *bin*HDOT in the presence of malicious adversaries.

Proof: The security of the protocol is proved in the hybrid model, assuming that the COTCD and OPE primitives, as well as the zero-knowledge proof of knowledge of Δ used in the protocol, are performed by a trusted oracle (or trusted party). This assumption is justified since we describe in Appendix an implementation of the COTCD protocol which is fully simulatable secure against malicious adversaries, and since a protocol for OPE, with similar security, was presented by Hazay and Lindell (2008). The security of the ZK proof of knowledge of Δ is based on standard arguments. All these protocols (COTCD, OPE and ZK proof) have security proofs based on the DCR assumption.

We compare the execution of the protocol between $\mathcal{P}_1$ and $\mathcal{P}_2$ to an execution with a *TTP*, where the TTP receives the inputs of both parties and computes the following functionality: If the input of $\mathcal{P}_1$ is w and the input of $\mathcal{P}_2$ is $\langle w', (Z_0, \ldots, Z_\ell)\rangle$, then the output of $\mathcal{P}_1$ is $Z_{d_h(w,w')}$. Otherwise if the input of $\mathcal{P}_1$ is a special symbol ρ then the output of $\mathcal{P}_1$ is a random value. Otherwise if the input of either party is a special symbol $\perp$ then the protocol terminates.

We first prove security in the case that $\mathcal{P}_1$ is corrupt and then in the case that $\mathcal{P}_2$ is corrupt.

$\mathcal{P}_1$ *is corrupt*. The idea behind the proof is that $\mathcal{P}_1$'s choices in the COTCD protocols define its input w. $\mathcal{P}_1$ is then supposed to sum the values it receives in the COTCD invocations and uses the result as its input to the OPE protocol. If it uses a different input to the OPE protocol, then, since it does not know Δ, it happens with overwhelming probability that $\mathcal{P}_1$ queries a value of the polynomial at a point which was not defined by $Z_0, \ldots, Z_\ell$ and receives a random answer.

More formally, let $\mathcal{A}$ be an adversary controlling $\mathcal{P}_1$. We construct a simulator Sim that generates the view of both parties, $\mathcal{A}$ and $\mathcal{P}_2$, in the hybrid model, given only access to $\mathcal{A}$ and to the ideal model:

1 Sim chooses a random value Δ and sends to $\mathcal{A}$ a commitment of Δ; Then, Sim runs the zero-knowledge proof of knowledge of Δ, with $\mathcal{A}$ as the verifier.

2 For each invocation of COTCD, Sim (simulating a trusted oracle that executes the COTCD protocol) receives $\mathcal{A}$'s bit input. It defines $\mathcal{A}$'s corresponding input bit w_i to be this value, and then chooses a random value t_i and sends it to $\mathcal{A}$ as the result of the COTCD protocol. (This happens if $\mathcal{A}$'s input to the protocol is 0 or 1. Otherwise, if $\mathcal{A}$'s input is $\perp$ or a value that is different from 0 or 1 then Sim terminates the protocol.)

   After finishing all executions of the COTCD protocol, Sim computes $\sigma_t = \sum t_i$.. Also, Sim has $\mathcal{A}$'s input $w = (w_0, \ldots, w_{\ell-1})$. It sends it to the trusted party computing the *bin*HDOT functionality and receives from it the result Z.

3 Sim then plays a trusted oracle computing the OPE protocol. Sim receives $\mathcal{A}$'s input to the OPE: If $\mathcal{A}$'s input to the OPE is σ_t then Sim sends it the answer Z; Otherwise, if its input is ρ, then Sim sends $\mathcal{A}$ a random value; if the input is $\perp$ then Sim terminates the protocol.

4 Sim outputs whatever $\mathcal{A}$ outputs and halts.

We now show that the joint output distribution of $\mathcal{A}$ and $\mathcal{P}_2$ in the hybrid model protocol execution is indistinguishable from the output of Sim and $\mathcal{P}_2$ in the ideal world simulation.

We start with the case where $\mathcal{A}$ sends $\perp$ to Sim, i.e., terminates its running in the protocol. This could happen in any phase of the protocol and as in the hybrid model, where Sim terminates, the protocol also terminates and $\mathcal{A}$ does not learns any further information.

$\mathcal{A}$ and $\mathcal{P}_2$ invoke COTCD protocol, where in the hybrid model, $\mathcal{A}$ sends its input to COTCD. If $\mathcal{A}$ sends 0 or 1 then it learns the appropriate result. Otherwise, the protocol terminates.

Now $\mathcal{A}$ and $\mathcal{P}_2$ invoke OPE protocol, consider first the case that $\mathcal{A}$'s input to the OPE is equal to σ_t. In the hybrid model execution, this results in evaluating the polynomial with an input which is the sum of the answers received in the COTCD protocols, namely with an input $f(\sigma_t) = f(\sigma_r + \Delta \cdot d_H(w, w'))$, where $w = w_1, \ldots, w_\ell$ and each w_i is $\mathcal{A}$'s input to the i^{th} invocation of COTCD. The result is $Z_{d_H(w,w')}$, as is the result in the simulation.

Consider now the case that $\mathcal{A}$'s input to the OPE is different from σ_t. Note that $\mathcal{A}$ does not know Δ, which was chosen at random, and therefore with probability $1-\ell/|\mathbb{Z}_N|$ it evaluates the polynomial at a point which is different than σ_r, $\sigma_r + \Delta$, …, $\sigma_r + \ell\Delta$ In the hybrid model execution, this results in receiving an answer which is random and independent of Z_0, …, Z_ℓ. This also happens in the simulation.

$\mathcal{P}_2$ *is corrupt*. Let $\mathcal{A}$ be an adversary controlling $\mathcal{P}_2$. The proof is based on the following ideas:

1 Sim extracts the value of Δ from the zero-knowledge proof of knowledge that is proved by $\mathcal{A}$

2 Sim then learns the inputs that $\mathcal{A}$ uses in the COTCD invocations, and based on these values the simulator computes w' and σ_r

3 It also learns the coefficients of the polynomial $P(\cdot)$ which is $\mathcal{A}$'s input to the OPE, and can therefore compute $Z_0 = P(\sigma_r)$, …, $Z_\ell = P(\sigma_r + \ell\Delta)$

4 Finally, the simulator sends $\langle w', (Z_0, \ldots, Z_\ell)\rangle$ to the TTP.

In more detail, we construct a simulator Sim that generates the view of both parties, $\mathcal{P}_1$ and $\mathcal{P}_2$, given only $\mathcal{P}_2$'s input in the ideal model.

1 Sim receives from $\mathcal{A}$ the commitment to Δ, and then plays the verifier in the zero-knowledge proof of knowledge. If Sim accepts the proof, it runs the knowledge extractor in order to learn Δ. Otherwise it terminates the execution of the protocol and sends ⊥ to the trusted party. (Also, here and throughout the simulation, if $\mathcal{A}$ sends ⊥ as input then Sim halts the execution and sends ⊥ to the trusted party.)

2 For each execution of COTCD, Sim acts as a trusted oracle that performs the protocol. Sim therefore learns $\mathcal{A}$'s input (x_i^0, x_i^1). First, Sim verifies that $|x_i^0 - x_i^1| = \Delta$ and if this property does not hold it aborts the protocol. Then, Sim defines each letter w_i' of the input word w' of $\mathcal{A}$:

- $w_i' = 1$, if $x_i^0 = x_i^1 + \Delta$
- $w_i' = 0$, Otherwise.

Finally, Sim computes $\sigma_r = \sum x_i^{w_i'} = \sum t_i^0$.

3 In the OPE step, Sim simulates a trusted party computing the OPE functionality. In this functionality, $\mathcal{A}$ provides an input but receives no output. It receives from $\mathcal{A}$ its input $P(\cdot)$ to the OPE, which could be a random polynomial. It then computes $Z_0 = P(f(\sigma_r))$, $Z_1 = P(f(\sigma_r + \Delta))$, …, $Z_\ell = P(f(\sigma_r + \ell\Delta))$. Now, Sim sends to the trusted party (computing *bin*HDOT) the input $\langle w', (Z_0, \ldots, Z_\ell)\rangle$.

4 Sim outputs whatever $\mathcal{A}$ outputs and halts.

Also here, we show that the joint output distribution of $\mathcal{A}$ and $\mathcal{P}_1$ in the hybrid model protocol execution is indistinguishable from the output of Sim and $\mathcal{P}_1$ in the ideal world simulation.

In the case where $\mathcal{A}$ sends ⊥ to Sim, i.e., terminates its running in the protocol, which could happen in any phase of the protocol, Sim terminates, as in real execution where the protocol terminates.

$\mathcal{A}$ and $\mathcal{P}_2$ run ℓ invocations of COTCD, where in the hybrid model are executed by TTP. $\mathcal{A}$ is enforced to commit Δ and proofs its knowledge as in the simulation, in addition, both values, (x_i^0, x_i^1), that $\mathcal{A}$ sends to the TTP satisfies $|x_i^0 - x_i^1| = \Delta$ and $\mathcal{P}_1$ learns one of the results, otherwise, the protocol aborts.

In the last step, $\mathcal{A}$ and $\mathcal{P}_1$ invoke OPE, where $\mathcal{P}_1$'s input is $\sigma_t = \sum t_i^0 + \Delta \cdot d_{H(w,w')}$ and $\mathcal{A}$ builds a polynomial of ℓ + 1 degree, this polynomial can be a random polynomial, and both parties sends their inputs to the TTP, as in the simulation, $\mathcal{P}_1$ learns the result and $\mathcal{A}$ does not learn any information. □

Efficiency. The overhead of the protocol is composed of running ℓ invocations of the COTCD protocol (which can be run in parallel, since the protocol is UC-secure), and a single invocation of the OPE protocol of Hazay and Lindell (2008).

The COTCD protocol, based on Jarecki and Shmatikov (2007), requires $O(1)$ rounds of communication and a constant number of exponentiations per party, including our auxiliary input and verification steps. In addition, the OPE protocol (Hazay and Lindell, 2008) requires $O(\ell + s)$ exponentiations, where s is a statistical security parameter, and a constant number of communication of rounds.

Thus, the overhead of the entire protocol is $O(\ell)$ rounds of communication and $O(\ell + s)$ exponentiations.

5.1 Securing the applications against malicious adversaries

The protocol described above is secure against malicious behaviour of either party. However, it does not enforce any structure of the inputs Z_0, …, Z_ℓ of $\mathcal{P}_2$ and therefore a corrupt $\mathcal{P}_2$ can set these inputs to arbitrary values. This 'feature' does not affect plain usage of the protocol, but it means that security against malicious adversaries cannot be guaranteed if the protocol is used for computing any functionality that requires specific relations between the Z_i values. Unfortunately, this is relevant to the relations required in the applications detailed in Section 3.1. For example, the *EQ* application, i.e., equality-based transfer, requires that $Z_1 = Z_2 = \cdots = Z_\ell$ (since all these values correspond to the case that $w \neq w'$). As a result, the protocol cannot be used ‘as is’ as a building block for protocols (secure against malicious adversaries) for the HDOT functionality for arbitrary alphabets, or for the *EQ* functionality.

In order to adapt the protocol for these tasks, it is required to add zero-knowledge proofs which assure $\mathcal{P}_1$ that the Z_i inputs follow the desired structure. This is of course possible in principle, but in this work we have not examined how to optimise the efficiently of such proofs. We will only describe here the steps which are required in order to design and implement an *EQ* protocol secure against malicious adversaries (protocols for the other applications can be designed in a similar way):

1 The protocol needs an additional step where $\mathcal{P}_1$ obtains a commitment $\mathsf{Com}(\sigma_r)$ to the base value $\sigma_r = \sum t_i^0$. This commitment can be computed given the commitments that $\mathcal{P}_2$ generates in the committed OT protocols; the correctness of the committed value can be proved using $\mathcal{P}_2$’s proofs about the Δ differences of its input pairs. (Namely, $\mathcal{P}_2$ must prove that there exist bits $b_0, \ldots, b_{\ell-1}$ such that $\sum x_i^{b_i} = \sigma_r$, and that $\forall i$ $x_i^1 = x_i^0 + \Delta$.)

2 The parties need to use a ‘committed OPE’ protocol, where $\mathcal{P}_2$ commits to the coefficients of its polynomial (such a protocol has not been described yet, but it is not hard to imagine how to implement it using techniques similar to those used for committed OT).

3 $\mathcal{P}_2$ must prove that there are values s; d such that s is committed to in $\mathsf{Com}(\sigma_r)$, d is committed to in $\mathsf{Com}(\Delta)$, and it holds that $P(s + d) = P(s + 2d) = \cdots = P(i + \ell d)$. The main challenge in designing this step is that $P(s + d)$ is computed to by multiplying the committed coefficients of P by powers of the value $s + d$. Namely, the proof is about the sum of multiplications of committed values.

6 Application: *m*-point SPIR

Another application of the HDOT protocol is a new variant of SPIR which we denote as *m*-point-SPIR. A definition and a discussion of single server PIR and symmetric PIR appear in, e.g., Kushilevitz and Ostrovsky (1997), and Cachin et al. (1999). In short, a PIR protocol involves a server with a database of N items $x_0, \ldots, x_{N-1}$ and a client who is interested in learning entry x_i of the database. This must be accomplished with $o(N)$ communication, without revealing i to the server, and (in the case of *symmetric* PIR) without revealing to the client anything but x_i.

The *m*-point-SPIR protocol that we define can be applied if at most m of the items of the server’s database have specific values, and all other items have some default value $\bar{x}$. The client must not know whether the value it learns is the default value $\bar{x}$ or one of the unique values. We describe below a couple of applications of *m*-point-SPIR. The *m*-point-SPIR functionality is similar to a simpler functionality, where the client learns a *random* value if its input does not match any of the m indices which have specific values. The latter functionality is much simpler to implement (using OPE), as we detail below.

We show a protocol which implements m-point-SPIR with $O(m\log N)$ communication and $O(m\log N)$ computation (the smaller m is, the more efficient the protocol is). Therefore, the communication is $o(N)$ as long as $m = o(N / \log N)$. Another nice property of the *m*-point-SPIR protocol is that it can be implemented based on the existence of oblivious transfer alone. This property is not known for general SPIR protocols. [Furthermore, it is known that there cannot exist any transparent black-box reduction of PIR to OT (Meier and Przydatek, 2006).]

The *m*-point-SPIR functionality is defined in the following way. The server has inputs $0 \le p_1, \ldots, p_m \le N - 1$, which are all distinct, and additional values $\bar{x}$, $x_{p_1}, \ldots, x_{p_m}$. The client has an input $0 \le i \le N - 1$. The output of the client is x_{p_j} if there is an index $1 \le j \le$ m such that $i = p_j$, or $\bar{x}$ if no such p_j exists.

1-point SPIR. The implementation of 1-point-SPIR is straightforward given our previous protocols. The parties simply execute the protocol $EQ_{x_{p_1},\bar{x}}(i, p_1)$, whose output is x_{p_1} if $i = p_1$, and $\bar{x}$ otherwise. (The *EQ* protocol is defined in Section 3.1.) The communication overhead is of the order of the length of the index i, namely $O(\log N)$, times the length of the security parameter (i.e., the length of the homomorphic encryption). (This is under the reasonable assumption that the length of the database values (the x values) is in the order of the length of the security parameter; otherwise the communication is $O(\log N \cdot |x|)$.) The computation overhead is $O(\log N)$, and it is composed of $O(\log N)$ homomorphic encryptions and $O(\log \log N)$ OTs.

m-point-SPIR. For the general case of *m*-point-SPIR, the server first defines m random values $z_1', \ldots, z_m'$ under the constraint that their exclusive-or is $\bar{x}$. It then defines values $z_1, z_2, \ldots, z_m$ satisfying the constraints

$$\begin{aligned} z_1 \oplus z_2' \oplus z_3' \oplus \cdots \oplus z_m' &= x_1 \\ z_1' \oplus z_2 \oplus z_3' \oplus \cdots \oplus z_m' &= x_2 \\ &\vdots \\ z_1' \oplus \cdots \oplus z_{m-1}' \oplus z_m &= x_m \end{aligned}$$

The parties execute the protocols $EQ_{z_1,z_1'}(i, p_1)$, $EQ_{z_2,z_2'}$ (i, p_2), up to $EQ_{z_m,z_m'}(i, p_m)$. The client then computes the exclusive-or of the m values that it learned in these protocols.

Correctness follows from the fact that if there exists a j coordinate for which $i = p_j$ then the client learns a single z_j value. Otherwise $i \ne p_1, \ldots, p_m$ and the client learns only z_j' values. Therefore, the exclusive-or of all the values that the client receives is equal to x_j in the former case, or to $\bar{x}$ in the latter case.

It is easy to verify the security of this protocol (assuming that the parties are semi-honest). Note that the client always performs the same operations and does not recognise whether it learned the value $\bar{x}$ or one of the m special values. The communication overhead is $O(m\log N)$ times the length of the security parameter, and the computation overhead is also $O(m\log N)$. This is therefore a SPIR protocol (with $o(N)$ communication) as long as $m = o(N / \log N)$, and in that case the computation overhead is also $o(N)$. (A 'traditional' PIR protocol will have $O(N)$ computation overhead, since it must also process the entries with the default value.)

Basing m-point-SPIR on OT. The *EQ* protocol (which is essentially the HDOT protocol) is based on using a homomorphic encryption scheme and an oblivious transfer. However, it is easy to see that the usage of homomorphic encryption can be replaced with the usage of oblivious transfer alone (as is done in the HDOT protocol for the malicious case). As a result, *m*-point-SPIR can be based on oblivious transfer alone.

Comparison to other protocols. Our *m*-point-SPIR protocol can be compared to OPE, in which the server has an $(m - 1)$-degree polynomial P, defined over a field of size at least N, and where the polynomial satisfies $P(p_j) = x_j$ for all $j \in [1, m]$. The client has input $0 \leq j \leq N - 1$ and it obliviously computes $P(j)$. The OPE protocol has communication and computation overheads of $O(m)$ field operations, but it has the drawback that for inputs not in $p_1, \ldots, p_m$ the client receives a random output rather than a specific value $\bar{x}$.

The *m*-point-SPIR protocol can also be compared to PIR protocols of the type of the protocol of Cachin et al. (1999) (that protocol is based on the ϕ-hiding assumption rather on general assumptions). These protocols, too, have the property that the server's work depends on the number of items in its database that have non-default values. Namely, it is $O(m)$ if the server has m items in its database, even if the range of the client's input is $[1, N]$. Still, in those protocols the sender is not able to set a 'default' value $\bar{x}$ to be returned for all other $N - m$ values of the client's input. Finally, the *m*-point-SPIR functionality can be implemented using Yao's generic protocol and a circuit of size $O(m\log N)$, and $m\log N$ invocations of OT. The observations in Section 3 comparing the overhead of the HDOT protocol to that of Yao's construction, are relevant in this case, too. We also believe that it is simpler to implement the *m*-point-SPIR protocol compared to implementing a circuit-based solution.

Application I: private matching for cardinality threshold. This is an example where it is important that $\mathcal{P}_1$ receives the default value if no match is found. The scenario involves two parties with private sets of *m* items, which want to find out if the size of the intersection of the sets is greater than some thresholds. The problem was defined in Freedman et al. (2004) as a variant of the private matching protocol which was the main subject of that paper. The solution there requires the parties to run an OPE for each item x_i of the first party, in which the first party either learns a specific value or a random value, depending on whether x_i is in the set of the second party. The parties then use Yao's protocol to evaluate a circuit whose input is the values learned by $\mathcal{P}_1$, and which computes whether the size of the intersection is greater than the threshold. We can use the *m*-point-SPIR protocol to replace the OPE: Suppose that $\mathcal{P}_1$'s inputs are $x_1, \ldots, x_n$ and $\mathcal{P}_2$'s inputs are $y_1, \ldots, y_n$. Then for each x_i the parties run an m-point SPIR where $\mathcal{P}_1$ learns α_i if $x_i \in \{y_1, \ldots, y_n\}$, or $\alpha_i + 1$ otherwise, where α is a random number chosen by $\mathcal{P}_2$. We can then ask $\mathcal{P}_1$ to sum the values it learned, and replace Yao's protocol with an OT_1^m, as was done in the *bin*HDOT protocol of Section 4.1. (This was impossible when an OPE was used, since in that case the sum was random if there was even a single item of $\mathcal{P}_1$ which was not in $\mathcal{P}_2$'s set.)

Application II: lottery service. As an example of another application of m-point-SPIR, consider a lottery service where the server has a range of tickets, only a few of which are winning tickets. The client uses the protocol to 'buy' a ticket, but the client must not know, at least not until some time in the future, whether this is a winning ticket. The server's database contains the prize corresponding to each winning ticket, or the default 'no prize' value $\bar{x}$ (which, of course, is associated to most of the tickets). It must be ensured that a client that receives the value $\bar{x}$ cannot identify that this is the default value. The server must not learn which ticket was chosen by the buyer. (A lottery service with many clients must handle many other different issues which we do not describe, but m-point-SPIR seems like a good approach for handling the purchase of tickets.)

7 Privacy-preserving computation of document similarity

HDOT protocols can be used to decide, in a secure way, whether two documents are similar (but not necessarily identical). More specifically, we consider the problem of two parties, each having a set of documents, that wish to find similar documents while ensuring that no party reveals any unnecessary data.

Motivation. An example of the need for privacy-preserving computation of similarity is the challenge facing conference committees that wish to detect the simultaneous submission of the same paper (or close variants of it) to more than a single conference. This practice is unacceptable, and conference committees attempt to identify parallel submissions, but they are hindered by the fact that papers must be handled confidentially and therefore conference committees cannot disclose the papers they have received to other committees.

If there was a *TTP* which was trusted by different conference committees then the problem could have been solved by the committees sending the documents to the TTP, which could then check them for similarity. Our goal is to build a privacy-preserving similarity algorithms that is run by the parties themselves and simulates the privacy offered by the TTP.

Algorithms for computing similarity between two documents were suggested by Broder et al. (1997) (and subsequent work) and are based on computing the Jaccard measure. The algorithms extract the words of each document, and sample them using Min-wise hashing (Broder et al., 2000) to create a set of words representing the original document. They then compare the sampled sets of the two documents. Similarity is defined as the size of the intersection of the sets of sampled words divided by the size of their union. Note that simple adversarial transformations to the documents, such as reordering words or adding some text, do not substantially affect the result of this class of algorithms.

7.1 Preliminaries

In this section, we introduce the similarity algorithms, and the cryptographic tools and notions used in our protocols. We begin by defining Rabin's fingerprinting scheme and Min-wise hash functions. It is important to emphasise that in this application we focus only on the case of semi-honest adversaries.

7.1.1 Rabin's fingerprinting scheme

Rabin's fingerprinting scheme (Broder, 1993; Rabin, 1981) is a method for mapping large objects to short tags. It is based on arithmetic modulo an irreducible polynomial, $P(\cdot)$. A fingerprinting family $\mathcal{F} = \{f : \Omega \to \{0,1\}^k\}$ (where Ω is a set of objects), fulfils the following two properties:

a if $f(A) \neq f(B)$ then $A \neq B$

b $\Pr[f(A) = f(B) \mid A \neq B] \approx 1/2^{O(K)}$, for $f \in_R \mathcal{F}$.

Rabin's fingerprinting algorithm has an efficient implementation over the field $GF(2^k)$, requiring a constant amount of memory and a linear computation overhead (see Broder, 1993; Rabin, 1981) for details.

7.1.2 Min-wise hash functions

The similarity algorithm uses sampling based on *Min-wise independent* permutations (Broder et al., 2000). Briefly, a set of permutations $\Pi \subseteq S_n$ is *Min-wise independent* if for any set $X \subseteq [n]$ and any $x \in X$, when π is chosen at random from Π it holds that

$$\texttt{Pr}\left(\texttt{Min}\{\pi(X)\} = \pi(x)\right) = \frac{1}{|X|}.$$

Namely, the probability that an element becomes the minimum element of the image of X under π is equal to all elements in X.

In practice, one can approximate the usage of Min-wise independent permutations by using pair-wise independent linear hash functions of the form $\pi(x) = ax + b$ (where a, b are chosen at random and $a \neq 0$). These functions are easy to represent and are efficient to calculate, and, as claimed by Broder et al. (2000) they perform well for practical applications of document similarity.

7.1.3 Computing similarity

Exact definitions of similarity between documents were given by Broder (2000), who investigated this problem for an application of clustering web pages. There defined the *resemblance* between documents, which is a number between 0 and 1. A resemblance close to 1 indicates that the two documents are 'roughly the same'.

A first step in computing similarity is representing each document $\mathcal{D}$ by a set of *shingles* $S(\mathcal{D})$. Shingles are unique sequences of tokens (which could be letters, words, lines, etc.) in a document, that are grouped into overlapping sets (Broder, 2000). Usually, all shingles have the same length; for instance, if we define each token to be a word, the four-shingling of the document $\mathcal{D} =$ (`a`, `rose`, `is`, `a`, `rose`, `is`, `a`, `rose`) is the set $S(\mathcal{D}) =$ {(`a`, `rose`, `is`, `a`), (`rose`, `is`, `a`, `rose`), (`is`, `a`, `rose`, `is`)}. [Shingling can also be defined in other ways (Broder, 1997).]

Broder (2000) defined the *resemblance* of two documents $\mathcal{D}_1$ and $\mathcal{D}_2$ (which is also known as the *Jaccard similarity coefficient*), as

$$r(\mathcal{D}_1, \mathcal{D}_2) = \frac{|S(\mathcal{D}_1) \cap S(\mathcal{D}_2)|}{|S(\mathcal{D}_1) \cup S(\mathcal{D}_2)|} \tag{1}$$

$r(\mathcal{D}_1, \mathcal{D}_2)$ measures the common features of both documents by computing the size of the intersection of their two sets of shingles divided by the size of the union of these sets. Intuitively, the resemblance captures the degree to which the two documents are similar. Notice that if two documents are identical, $\mathcal{D}_1 = \mathcal{D}_2$, then $r(\mathcal{D}_1, \mathcal{D}_2) = 1$, and that if two documents are totally different, $\mathcal{D}_1 \cap \mathcal{D}_2 = \phi$, then $r(\mathcal{D}_1, \mathcal{D}_2) = 0$.

To simplify the computation, it is common to map shingles into shorter, fixed-length numerical values using Rabin's fingerprinting algorithm (Rabin, 1981; Broder, 1993) and apply the rest of the computation to these values.

The process computing the resemblance uses the entire document, this process is inefficient because it requires large amounts of memory and runtime. Therefore, Broder et al. suggested to improve the computation of similarity by first *sampling* part of the shingles, using *Min-wise hashing* functions (Broder, 1997, 2000; Broder et al., 2000), and then computing the function $r(\cdot,\cdot)$ of the sampled values. Two different ways were suggested for sampling shingles of a document:

- A single permutation is used to sample a subset of the shingles of each document in the following way: each shingle is mapped to a value by the permutation π, and from each document we take the shingles that were mapped to the n smallest values. The function $r(\cdot,\cdot)$ is then applied to these samples. [This algorithm is detailed and analysed in Broder et al. (1997).]

- Similarity can also be computed using multiple permutations. Each permutation is used to choose the shingle from each document that is mapped by it to the smallest value. For each permutation, the two values that are chosen by it from the two documents are compared. We use this method, as is detailed and justified in the text below.

Computing similarity using multiple permutations. We assume that shingles are represented by values in a range of size p. The computation of similarity uses a set of n *permutations* $\{\pi_0, \pi_1, \ldots, \pi_{n-1}\}$ chosen uniformly over the set of Min-wise independent permutations of $[p]$. Computing similarity operates in the following way:

1 For each document, the *minimall* according to each permutation is sampled. Namely, the parties compute $\texttt{Min}[\pi_i(S(\mathcal{D}_1))]$ and $\texttt{Min}[\pi_i(S(\mathcal{D}_2))]$, where $i \in \{0, \ldots, n-1\}$ and where Min outputs the minimall value of the set of its inputs.

2 The value $\psi(\mathcal{D}_1, \mathcal{D}_2)$ is defined to be the number of elements for which $\texttt{Min}[\pi_i(S(\mathcal{D}_1))] = \texttt{Min}[\pi_i(S(\mathcal{D}_2))]$.

3 The resemblance is defined as $\psi(\mathcal{D}_1, \mathcal{D}_2)/n$.

It is easy to see Broder (2000), that

$$\texttt{Pr}_\pi\left(\texttt{Min}\{\pi(S(\mathcal{D}_1))\} = \texttt{Min}\{\pi(S(\mathcal{D}_2))\}\right) = \frac{|S(\mathcal{D}_1) \cap S(\mathcal{D}_2)|}{|S(\mathcal{D}_1) \cup S(\mathcal{D}_2)|} = r(\mathcal{D}_1, \mathcal{D}_2) \quad (2)$$

Therefore, the expected value of $\psi(\mathcal{D}_1, \mathcal{D}_2)/n$ is $r(\mathcal{D}_1, \mathcal{D}_2)$, and it can be used to estimate this value. This provides a way for estimating the value of $r(\mathcal{D}_1, \mathcal{D}_2)$ (which does not require to compute the exact intersection of union of the documents). An analysis of the variance of this estimation appears in Broder (2000).

Our approach. We have chosen to use the multiple permutations approach because the accuracy of this solution has a smaller variance than that of the solution which is based on a single permutation. Another important reason that makes this solution preferable for our purposes is that it requires to compare the item sampled by a certain permutation from the first input which exactly one item, the item sampled by this same permutation from the second input. This property is useful for two reasons:

1 Implementing a privacy preserving version of the second solution is easier than implementing a similar variant of the first solution, since it requires checking the equality of pairs of items, rather than computing the intersection of larger sets.

2 In our last protocol each party first samples a set of items from its own input, and then the protocol uses only part of the sampled sets of both parties (without letting the parties know which items are used by the protocol). This is easier if we know that using the j^{th} element of the first set requires using the j^{th} element of the second set.

7.2 Problem statement

We consider a scenario with two parties: $\mathcal{P}_1$ which has document $\mathcal{D}_1$ and $\mathcal{P}_2$ which has document $\mathcal{D}_2$. Both parties must run a protocol that outputs 1 if $\mathcal{D}_1$ is similar to $\mathcal{D}_2$, and output 0 otherwise. This must be done without revealing any other information about the documents. We assume that two documents are similar if $r(\mathcal{A}, \mathcal{B})$ is greater some predefined threshold, which is a parameter set by the parties.

We will describe three *ideal scenarios* for checking similarity with a TTP. The first scenario does not leak any information except for the result of whether $r(\mathcal{A}, \mathcal{B})$ is greater than the threshold, while the second and third scenarios reveal some additional information. We will then describe two-party protocols that simulate these ideal scenarios – namely, do not leak more information than in the corresponding ideal scenario. (As can be anticipated, the protocols corresponding to the second and third scenarios will be more efficient than the protocol corresponding to the first scenario.)

Ideal scenario 1 – Naive TTP. In this scenario, the TTP receives both documents $\mathcal{D}_1$ and $\mathcal{D}_2$, and computes the similarity between them. The computation of the similarity is based on the Jaccard similarity [equation (1)], applied either to the entire documents, or to a sampling of the shingles of each document (for instance, by sampling the documents using multiple permutations and computing similarity based on the sampled values).

Ideal scenario 2 – TTP with sampling by the parties (TTP SbP). In this scenario, the sampling of the documents is done by the parties themselves: both $\mathcal{P}_1$ and $\mathcal{P}_2$ perform the sampling of their own documents and send the results to the TTP. The TTP then evaluates similarity by applying, to the sampled sets, the algorithm that computes similarity using multiple permutations (described in Section 7.1.3).

This approach is more efficient than ideal scenario 1, but it leaks some additional data since each party knows which values of its set were used in evaluating the similarity. (In the extreme case, if the size of the sampled subset is 1, then $r(\mathcal{D}_1, \mathcal{D}_2) = 1$ implies that the specific sampled shingle exists in both documents.)

Ideal scenario 3 – TTP with obscured sampling by the parties, i.e., TTP with obscured SbP. This scenario is similar to the previous one but it aims to somewhat obscure the exact sampled shingles that are used to compute the similarity. This is done in the following way:

1 Both parties sample $k \cdot n$ items from their documents (where $k > 1$ and n are parameters) and send the sampled items to the TTP.

2 The TTP chooses a random subset of n pairs of items from these subsets and uses it to evaluate similarity.

The usage of part of the sampled elements to compute similarity aims to improve the privacy of the protocol of scenario 2, since no party knows for sure what elements were used. However, some information does leak (compared to the first protocol were all items are used for computing similarity). It seems that a larger value of the parameter k corresponds to better privacy, but the exact privacy analysis is out of the scope of this paper.

7.3 Secure protocols

Secure protocols compute the functionality without revealing more information than is revealed in the ideal scenario. We first describe in brief a protocol for the first ideal scenario. We then focus on protocols for the second and third scenarios, since the first protocol requires large communication and computation overheads as the parties must apply cryptographic operations to the entire documents. Note that even Broder et al.'s insecure protocol is based on sampling the documents in order to reduce the overhead of the protocol.

We describe protocols which compute similarity between a pair documents. Section 7.4 discusses the comparison of two *sets* of documents, looking for any pair of similar documents that appear in both sets. Instead of requiring the parties to compute $O(N^2)$ comparisons (for sets of N documents), it suggests a more efficient protocol, based on hashing into bins, which computes only $O(N)$ comparisons.

7.3.1 Protocols for ideal scenario 1

Using the full documents. If no sampling is done then the protocol must compare the complete two sets of shingles and output 1 if the size of their intersection is greater than some threshold. This is exactly the task computed by the *private matching for cardinality threshold* protocol of Freedman et al. (2004) and therefore we could simply apply this protocol to the shingle sets of the two parties. The protocol requires computing a constant number of homomorphic encryption operations for each shingle, and computing a 1-out-of-2 oblivious transfer for each bit of the representation of the shingle values (needed for computing a Yao circuit). This results in at least $m\log m$ OT_1^2s, for inputs of m shingles. This overhead is almost linear, but the size of the input here, m, is pretty large, since no sampling is applied to the documents.

It is possible to relax the privacy requirements of the protocol and enable it to output the *size* of the intersection instead of evaluating whether the size of intersection is larger than a threshold. In this case, the protocol can be implemented using a set intersection protocol, which computes the size of the intersection of two sets known to the two parties (rather than computing the cardinality threshold, which is a more complex operation). Two known protocols for set intersection are that of Huberman et al. (1999) [also described and analysed by Agrawal et al. (2003)], and that of Freedman et al. (2004). The former was proved to be secure only in the random oracle model, whereas the latter was proved in the standard model.

7.3.2 A protocol for ideal scenario 2 – sampling by parties

The protocol consists of a sampling step and a computation step. First, the parties agree on n Min-wise independent permutations. Then, each party samples its document by itself using these n permutations, as described in Section 7.1.3. The last step computes similarity, that is, outputs 1 if the number of equal pairs is at least τ (where $0 \le \tau \le n$ is a parameter). This is done by representing the output of the n permutations as an n-letter word, defined over an alphabet sufficiently large to contain the fingerprint of the sampled shingles. Then the threshold protocol, $\mathrm{HDOT}_{1,0}^{|W|-\tau}(W, W')$, is run, where W and W' are the words representing the sets of shingles sampled by the permutations from $\mathcal{D}_1$, $\mathcal{D}_2$, respectively, and the protocol outputs 1 if the Hamming distance is at most $|W| - \tau$. (This protocol is defined in Section 4.1 as one of the straightforward applications of HDOT.)

7.3.3 A protocol for scenario 3 – obscured sampling by the parties

As was discussed earlier, the previous protocol simulates a setting where the TTP reveals to the parties the identities of the sampled values that are used for computing similarity. This privacy leakage might be somewhat reduced by letting each party sample first a large set of elements (which it knows), and then use a random subset of these elements for evaluating similarity while keeping this subset hidden from the other party. This approach is implemented by the protocol described in Figure 4. The protocol uses an additional parameter, k, ($k > 1$). Each party samples kn words from its set, but only n of these words take part in the final evaluation.

The protocol starts with each party sampling kn shingles from its document, where both parties use the same set of permutations for this task. Next, the parties compute together a random list of kn homomorphic encryptions of values $\alpha_0, \ldots, \alpha_{k\cdot n-1}$, of which n are encryptions of 1 and the rest are encryptions of 0 (but no single party knows which encryptions are of which value). They then execute an $EQ_{\mathcal{V}_0,\mathcal{V}_1}$ protocol (of Section 4.1) for each of the kn pairs of words. For each of these executions $\mathcal{P}_2$ chooses a random value r_i. $\mathcal{P}_1$ learns the value $\mathcal{V}_1 = r_i + \alpha_i$ if the words are equal, and if they are different it learns the value $\mathcal{V}_0 = r_i$. As a result, the equality of input words only affects the results of the n pairs which correspond to the α_i values which equal 1. The two parties then run an oblivious transfer protocol (similar to the one used in the last step of HDOT protocol) to compute the output of the protocol. The protocol is detailed in Figure 4.

The proof of correctness is similar to that of the HDOT protocol. The overhead of the protocol is about the same as that of the document similarity protocol for scenario 2, when that protocol is run with a sample size of $k \cdot n$. (Namely, the usage of a parameter $k > 1$ increases the overhead by a factor of k compared to the previous protocol. There is a smooth transition from the protocol of scenario 2, which corresponds to setting $k = 1$, to the protocol of scenario 3 which uses $k > 1$.) This observation was verified in the experiments we conducted, detailed in Section 7.5. More precisely, $\mathcal{P}_1$ performs $nk\ell$ homomorphic encryptions and nk decryptions, and $\mathcal{P}_2$ performs $2nk\ell$ encryptions. The parties also run $nk \log(\ell + 1)$ OT_1^2s. The communication consists of $O(nk\ell)$ encrypted items. The security analysis is similar to that of the previous protocol.

7.4 Comparing many documents

We have introduced protocols that compute similarity between two documents, but in many scenarios each party has a set of documents and the parties wish to identify any pair of similar documents (this is indeed the case of the problem encountered by the programme committees, that was the motivation of our research). Let us assume that each party (committee) has n documents to compare with the other party. A naive solution is to compare each pair of documents of the two parties and execute the similarity protocol n^2 times. We would like to reduce this overhead.

The number of executions of the similarity protocol can be indeed reduced if the parties compare only documents which are likely to be similar. This can be done by mapping documents to different 'bins' such that similar documents are mapped to the same bin by both parties. In this case, it is required only to compare the documents that are mapped to a certain bin by the first party with documents mapped to the same bin by the second party. Naturally, we must make sure that no information about the documents is leaked or revealed by the mapping to the bins.

Figure 4 The similarity protocol for ideal scenario 3

$\text{Similarity}_{\tau}^{k:n}\left(\langle w_0, w_1, ..., w_{kn-1}\rangle, \langle w'_0, w'_1, ..., w'_{kn-1}\rangle\right)$

Input: $\mathcal{P}_1$ has an input $\langle w_0, w_1, \ldots, w_{kn-1}\rangle$; $\mathcal{P}_2$ has an input $\langle w'_0, w'_1, ..., w'_{kn-1}\rangle$, and both parties agree on a threshold τ.

Output: The protocol chooses at random a set of n of the kn words. $\mathcal{P}_1$ receives 1 if the number of equal words among these is at least τ, otherwise it receives 0; $\mathcal{P}_2$ does not learn any output.

$\mathcal{P}_1$ knows the decryption key to a homomorphic encryption scheme E_{pk}.

1. $\mathcal{P}_1$ sends to $\mathcal{P}_2$ n homomorphic encryptions of 1, and $(k - 1)n$ homomorphic encryptions of 0, ordered randomly.
2. $\mathcal{P}_2$ receives these kn encryptions and randomises their order. Call the resulting encryptions $E_{pk}(\alpha_0), \ldots, E_{pk}(\alpha_{kn-1})$.

   Note that these two steps can be performed before the parties receive their inputs.
3. For $i \leftarrow 0$ to $(kn - 1)$

   $\mathcal{P}_2$ chooses a random value $r_i \in_R \mathcal{F}$, computes $E_{pk}(r_i)$ and $E_{pk}(r_i + \alpha_i)$, and both parties execute the protocol $EQ_{E_{pk}(r_i+\alpha_i),E_{pk}(r_i)}(w_i, w'_i)$. In this protocol $\mathcal{P}_1$ learns $E_{pk}(r_i + \alpha_i)$ if $w_i = w'_i$, and learns $E_{pk}(r_i)$ otherwise.

   At the end of this step, $\mathcal{P}_1$ obtains the values $\{E(\beta_0), \ldots, E(\beta_{kn-1})\}$, where

   $$\beta_i = \begin{cases} r_{i,} & w_i \neq w'_i \quad \text{or} \quad \alpha_i = 0 \\ r_{i+1}, & w_i = w'_i \quad \text{and} \quad \alpha_i = 1 \end{cases}$$

   (Note that the equality of the inputs affects the value of β_i only when $\alpha_i = 1$.)
4. $\mathcal{P}_2$ sums the r values modulo $n + 1$ to obtain $\sigma_r = \sum_0^{kn-1} r_i \bmod(n+1)$.
5. $\mathcal{P}_1$ sums the β values and decrypts the result to obtain $\sigma_\beta = \sum_0^{kn-1} \beta_i \bmod(n+1)$. (Note that this value is equal to σ_r plus the number of indices i in which $w_i = w'_i$ and $\alpha_i = 1$.)
6. Both parties execute an OT_1^{n+1} protocol with the following inputs:
   - $\mathcal{P}_1$ is the receiver and its input is $\sigma_\beta \bmod (n + 1)$.
   - $\mathcal{P}_2$ is the sender and its input is $\{X_0, \ldots, X_{\tau-1}, X_\tau, \ldots, X_{n-1}, X_n\}$ such that

   $$X_i = \begin{cases} 1, & \text{if } (i - \sigma_r)\bmod(n+1) \geq \tau \\ 0, & \text{Otherwise} \end{cases}$$

   $\mathcal{P}_1$ receives the result of the OT protocol; $\mathcal{P}_2$ does not receive any output.

To categorise the documents we search for distinctive properties that are likely to be the same for two version of a paper submitted to two conferences. In the programme committee example we can assume that the set of authors of a paper has not changed, and it is therefore possible to use an ordered list of the names of authors of each document as a distinctive property. (If we suspect that authors may add spurious names to their document, it is possible to use each author name as a separate index and map a document with ℓ authors to ℓ different bins.) The mapping to bins is performed using a random hash function with a range of size B, applied to the set of authors of each paper. It has been shown (Freedman et al., 2004) that if the hash function maps each item to a random bin there is a high probability (over the selection of the hash function) that each bin contains at most $M = n / B + \mathcal{O}(\sqrt{(n / B)\log B} + \log B)$ elements (see, e.g., Freedman et al., 2004).

After mapping the documents to the bins, both parties need only compare the documents that have been categorised to the same bin by both of them. It is important that no party learns the number of the documents in each of the bins of the other party; to ensure this, each party must add several random documents to each of its bins such that the number of documents in each bin is exactly $M = n / B + \mathcal{O}(\sqrt{(n / B)\log B} + \log B)$.

Overhead. After mapping the documents to bins, the parties need to compare the documents that are mapped to the same bin by both parties. Therefore, the total number of comparisons between documents is $B \cdot M^2 = B \cdot (n / B + \mathcal{O}(\sqrt{(n / B)\log B} + \log B))^2$. If we choose B to be n / log n, the similarity protocol is executed $\mathcal{O}(n \log n)$ times. This analysis is asymptotic. For specific values of n, the parties should search for the value of B that produces the best overhead.

7.5 Implementation and experiments

Configuration. We implemented the document similarity protocol using Java 1.5. The experiments used the following settings:

1 Homomorphic encryption was done using Paillier's method, with a ring Z_n of size 1,024 bits

2 the sizes of the parameters p and q for the OT protocol [based on the Bellare-Micali construction (Bellare and Micali, 1990)] were 1,024 and 160 bits, respectively

3 the shingle size was seven letters

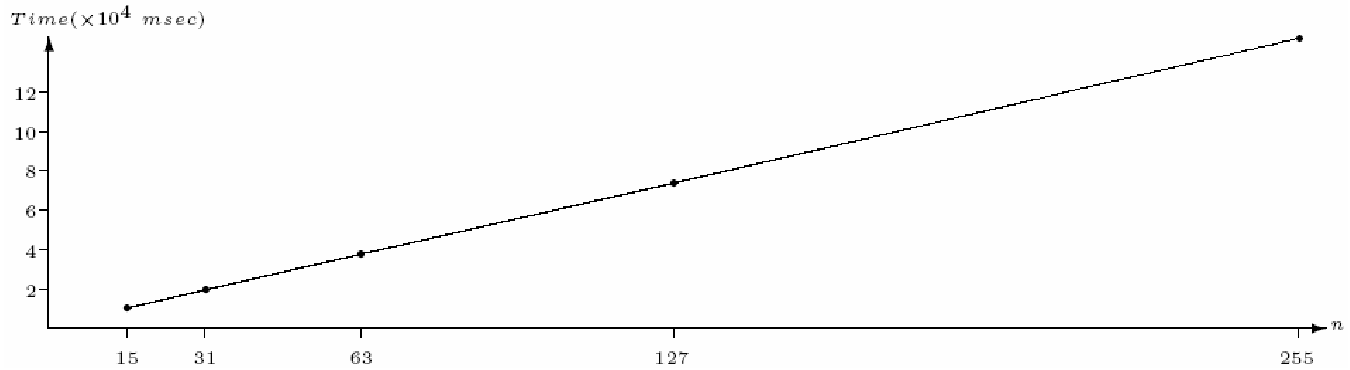
4 we used Rabin's algorithm to generate a fingerprint of 32 bits, but only 31 bits were utilised (the ideal size of the words should be $2^d - 1$, for any d).

The experiments were performed using two machines, each with a 2.8 GHz Pentium D processors and 1 GB of RAM, running the Linux OS.

Results. We used two PDF files, with a similarity of about 85%. Each file contained about 9,500 words in 35 pages. Sampling was performed where the sample size was n = 15, 31, 63, …, 255, and the privacy parameter was k = 1, 2, …, 8.

Figure 5 shows the runtime of the protocol for ideal scenario 2 (where k = 1, or running HDOT protocol). The points represent the total runtime of the protocol, as we can see, the graph is linear in the size of the sample n. Figure 6 presents the runtime of the protocol for ideal scenario 3 (for n = 255 and k = 1, …, 8), where the bars represent the runtime spent on running *bin*HDOT invocations, namely, runtime spent on comparing binary words, and the points represent the total runtime spent in each execution of the protocol. The graph is linear in k and demonstrates that most of the runtime is spent on comparing words (note that the bars are very close to the line.) Both graphs agree with the observation that the runtime is linear in the size of the samples.

Figure 5 Run time of HDOT



Analysing the runtime of the different parts of the protocol reveals that, on average, comparing two 31-bit words ($\ell = 31$) took 568 msec, where 24 msec were spent counting equal bits and 540 msec were spent on the OT protocol. The first item corresponds to ℓ homomorphic additions, and the second to $5 = \log(\ell + 1)$ OTs executed one after the other, with an average time of about 110 msec per OT. In the preprocessing step, each homomorphic encryption took about 7msec. This observation shows that the overhead of OT (which involves communication between the parties) is much larger than that of a homomorphic encryption. Note also that with a running time of about 0.56 sec for comparing every pair of words, the overall running time of the protocol, which compares a few hundred words, is reasonable, although not instantaneous.

Acknowledgements

We would like to thank Gil Segev and Kobbi Nissim for the helpful comments they provided on an earlier version of this work.

Supported by the SFEROT project funded by the European Research Council (ERC).

References

Agrawal, R., Evfimievski, A.V. and Srikant, R. (2003) 'Information sharing across private databases', in Halevy, A.Y., Ives, Z.G. and Doan, A. (Eds.): *SIGMOD Conference*, ACM, pp.86–97.

Aiello, W., Ishai, Y. and Reingold, O. (2001) 'Priced oblivious transfer: how to sell digital goods', in *Advances in Cryptology – Eurocrypt '01*, pp.119–135, Springer-Verlag, London, UK.

Beaver, D. (1996) 'Correlated pseudorandomness and the complexity of private computations', in *STOC*, pp.479–488.

Bellare, M. and Micali, S. (1990) 'Non-interactive oblivious transfer and applications', in *Advances in Cryptology – Crypto '89*, pp.547–557, Springer-Verlag, London, UK.

Ben-David, A., Pinkas, B. and Nisan, N. (2008) 'Fairplaymp – a system for secure multi-party computation', in *ACM Conference on Computer and Communications Security – ACM CCS 2008*, ACM, October.

Ben-Or, M., Goldwasser, S. and Wigderson, A. (1988) 'Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract)', in *STOC*, pp.1–10, ACM.

Blackburn, S., Blake-Wilson, S., Burmester, M. and Galbraith, S. (1998) 'Shared generation of shared RSA keys', University of Waterloo Technical report, CORR, pp.98–19.

Blake, I.F. and Kolesnikov, V. (2006) 'Conditional encrypted mapping and comparing encrypted numbers', in Crescenzo, G.D. and Rubin, A.D. (Eds.): *Financial Cryptography and Data Security, 10th International Conference, FC 2006*, Anguilla, British West Indies, Revised Selected Papers, Lecture Notes in Computer Science, Springer, 27 February–2 March, Vol. 4107, pp.206–220.

Bogetoft, P., Damgård, I., Jakobsen, T., Nielsen, K., Pagter, J. and Toft, T. (2006) 'A practical implementation of secure auctions based on multiparty integer computation', in Crescenzo, G.D. and Rubin, A.D. (Eds.): *Financial Cryptography and Data Security, 10th International Conference, FC 2006*, Anguilla, British West Indies, Revised Selected Papers, Lecture Notes in Computer Science, Springer, 27 February–2 March, Vol. 4107, pp.142–147.

Boneh, D. (Ed.) (2003) *Advances in Cryptology – CRYPTO 2003, Proceedings, Lecture Notes in Computer Science*, Springer, Vol. 2729.

Broder, A.Z. (1993) *Some Applications of Rabin's Fingerprinting Method*, pp.143–152, Springer-Verlag.

Broder, A.Z. (1997) 'On the resemblance and containment of documents', in *SEQUENCES '97: Proceedings of the Compression and Complexity of Sequences 1997*, IEEE Computer Society, Washington, DC, USA, p.21.

Broder, A.Z. (2000) 'Identifying and filtering near-duplicate documents', in Giancarlo, R. and Sankoff, D. (Eds.): *Proceedings on Combinatorial Pattern Matching, 11th Annual Symposium, CPM 2000, Lecture Notes in Computer Science*, Montreal, Canada, Springer, 21–23 June, Vol. 1848, pp.1–10.

Broder, A.Z., Charikar, M., Frieze, A.M. and Mitzenmacher, M. (2000) 'Min-wise independent permutations', *J. Comput. Syst. Sci.*, Vol. 60, No. 3, pp.630–659.

Broder, A.Z., Glassman, S.C., Manasse, M.S. and Zweig, G. (1997) 'Syntactic clustering of the web', *Computer Networks*, Vol. 29, Nos. 8–13, pp.1157–1166.

Cachin, C., Micali, S. and Stadler, M. (1999) 'Computationally private information retrieval with polylogarithmic communication', in *EUROCRYPT*, pp.402–414.

Camenisch, J. and Shoup, V. (2003) 'Practical verifiable encryption and decryption of discrete logarithms', in Boneh, D. (Ed.): *Advances in Cryptology – CRYPTO 2003, Proceedings, Lecture Notes in Computer Science*, Springer, Vol. 2729, pp.126–144.

Camenisch, J., Neven, G. and Shelat, A. (2007) 'Simulatable adaptive oblivious transfer', in Naor, M. (Ed.): *Proceedings on Advances in Cryptology – EUROCRYPT 2007, Lecture Notes in Computer Science*, Barcelona, Spain, Springer, 20–24 May, Vol. 4515, pp.573–590.

Canetti, R. (2000) 'Security and composition of multiparty cryptographic protocols', *J. Cryptology*, Vol. 13, No. 1, pp.143–202.

Chang, Y-C. (2004) 'Single database private information retrieval with logarithmic communication', in Wang, H., Pieprzyk, J. and Varadharajan, V. (Eds.): *ACISP, Lecture Notes in Computer Science*, Vol. 3108, pp.50–61, Springer.

Charikar, M.S. (2002) 'Similarity estimation techniques from rounding algorithms', in *STOC '02: Proceedings of the Thirty-fourth Annual ACM Symposium on Theory of Computing*, ACM, New York, NY, USA, pp.380–388.

Cramer, R., Damgård, I. and Schoenmakers, B. (1994) 'Proofs of partial knowledge and simplified design of witness hiding protocols', in *CRYPTO '94: Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology*, Springer-Verlag, London, UK, pp.174–187.

Crescenzo, G.D. and Rubin, A.D. (Eds.) (2006) *Financial Cryptography and Data Security, 10th International Conference, FC 2006*, Anguilla, British West Indies, Revised Selected Papers, Lecture Notes in Computer Science, Springer, 27 February–2 March, Vol. 4107.

Damgård, I. and Jurik, M. (2001) 'A generalisation, a simplification and some applications of Paillier's probabilistic public-key system', in Kim, K. (Ed.): *Public Key Cryptography, Lecture Notes in Computer Science*, Vol. 1992, pp.119–136, Springer.

Even, S., Goldreich, O. and Lempel, A. (1982) 'A randomized protocol for signing contracts', in *Advances in Cryptology – Crypto '82*, pp.205–210.

Fagin, R., Naor, M. and Winkler, P. (1996) 'Comparing information without leaking it', *Communications of the ACM*, Vol. 39, No. 5, pp.77–85.

Feigenbaum, J., Ishai, Y., Malkin, T., Nissim, K., Strauss, M.J. and Wright, R.N. (2006) 'Secure multiparty computation of approximations', *ACM Transactions on Algorithms*, Vol. 2, No. 3, pp.435–472.

Freedman, M.J., Nissim, K. and Pinkas, B. (2004) 'Efficient private matching and set intersection', in Cachin, C. and Camenisch, J. (Eds.): *EUROCRYPT, Lecture Notes in Computer Science*, Vol. 3027, pp.1–19, Springer.

Gentry, C. and Ramzan, Z. (2005) 'Single-database private information retrieval with constant communication rate', in Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C. and Yung, M. (Eds.): *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lecture Notes in Computer Science*, Vol. 3580, pp.803–815, Springer.

Gertner, Y., Ishai, Y., Kushilevitz, E. and Malkin, T. (1998) 'Protecting data privacy in private information retrieval schemes', in *STOC '98: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, ACM, New York, NY, USA, pp.151–160.

Giancarlo, R. and Sankoff, D. (Eds.) (2000) *Proceedings on Combinatorial Pattern Matching, 11th Annual Symposium, CPM 2000, Lecture Notes in Computer Science*, Montreal, Canada, Springer, 21–23 June, Vol. 1848.

Goethals, B., Laur, S., Lipmaa, H. and Mielikinen, T. (2004) 'On private scalar product computation for privacy-preserving data mining', in *Proc. of the Seventh Annual International Conference in Information Security and Cryptology, LNCS*, pp.104–120, Springer-Verlag.

Goldreich, O. (2004) *Foundations of Cryptography, Basic Applications*, Vol. 2, Cambridge University Press, New York, NY, USA.

Goldreich, O., Micali, S. and Wigderson, A. (1987) 'How to play any mental game or A completeness theorem for protocols with honest majority', in *Proceedings of the 19th Annual Symposium on Theory of Computing*, May, pp.218–229.

Green, M. and Hohenberger, S. (2007) 'Blind identity-based encryption and simulatable oblivious transfer', in *ASI-ACRYPT*, pp.265–282.

Hazay, C. and Lindell, Y. (2008) 'Efficient oblivious polynomial evaluation and transfer with simulation-based security', Manuscript.

Huberman, B.A., Franklin, M.K. and Hogg, T. (1999) 'Enhancing privacy and trust in electronic communities', in *ACM Conference on Electronic Commerce*, pp.78–86.

Indyk, P. and Woodruff, D.P. (2006) 'Polylogarithmic private approximations and efficient matching', in Halevi, S. and Rabin, T. (Eds.): *TCC, Lecture Notes in Computer Science*, Vol. 3876, pp.245–264, Springer.

Ishai, Y., Kilian, J., Nissim, K. and Petrank, E. (2003) 'Extending oblivious transfers efficiently', in Boneh, D. (Ed.): *Advances in Cryptology – CRYPTO 2003, Proceedings, Lecture Notes in Computer Science*, Springer, Vol. 2729, pp.145–161.

Jarecki, S. and Shmatikov, V. (2007) 'Efficient two-party secure computation on committed inputs', in Naor, M. (Ed.): *Proceedings on Advances in Cryptology – EUROCRYPT 2007, Lecture Notes in Computer Science*, Barcelona, Spain, Springer, 20–24 May, Vol. 4515, pp.97–114.

Kushilevitz, E. and Ostrovsky, R. (1997) 'Replication is not needed: single database, computationally-private information retrieval', in *FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97)*, IEEE Computer Society, Washington, DC, USA, p.364.

Lindell, Y. and Pinkas, B. (2007) 'An efficient protocol for secure two-party computation in the presence of malicious adversaries', in Naor, M. (Ed.): *Proceedings on Advances in Cryptology – EUROCRYPT 2007, Lecture Notes in Computer Science*, Barcelona, Spain, Springer, 20–24 May, Vol. 4515, pp.52–78.

Lindell, Y., Pinkas, B. and Smart, N.P. (2008) 'Implementing two-party computation efficiently with security against malicious adversaries', in Ostrovsky, R., Prisco, R.D. and Visconti, I. (Eds.): *SCN, Lecture Notes in Computer Science*, Vol. 5229, pp.2–20, Springer.

Lipmaa, H. (2005) 'An oblivious transfer protocol with log-squared communication', in Zhou, J., Lopez, J., Deng, R.H. and Bao, F. (Eds.): *The 8th Information Security Conference (ISC'05), Lecture Notes in Computer Science*, 20–23 September, Vol. 3650, pp.314–328, Springer-Verlag.

Malkhi, D., Nisan, N., Pinkas, B. and Sella, Y. (2004) 'Fairplay – secure two-party computation system', in *USENIX Security Symposium*, pp.287–302, USENIX.

Meier, R. and Przydatek, B. (2006) 'On robust combiners for private information retrieval and other primitives', in Dwork, C. (Ed.): *Advances in Cryptology – CRYPTO '06, Lecture Notes in Computer Science*, Vol. 4117, pp.555–569, Springer-Verlag, August.

Naor, M. (Ed.) (2007) *Proceedings on Advances in Cryptology – EUROCRYPT 2007, Lecture Notes in Computer Science*, Barcelona, Spain, Springer, 20–24 May, Vol. 4515.

Naor, M. and Nissim, K. (2001) 'Communication preserving protocols for secure function evaluation', in *STOC*, pp.590–599.

Naor, M. and Pinkas, B. (1999) 'Oblivious transfer and polynomial evaluation', in *STOC '99: Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, ACM, New York, NY, USA, pp.245–254.

Naor, M. and Pinkas, B. (2001) 'Efficient oblivious transfer protocols', in *SODA '01: Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp.448–457.

Naor, M. and Pinkas, B. (2005) 'Computationally secure oblivious transfer', *J. Cryptology*, Vol. 18, No. 1, pp.1–35.

Paillier, P. (1999) 'Public-key cryptosystems based on composite degree residuosity classes', in *EUROCRYPT*, pp.223–238.

Paillier, P. (2000) 'Trapdooring discrete logarithms on elliptic curves over rings', in Okamoto, T. (Ed.): *ASIACRYPT, Lecture Notes in Computer Science*, Vol. 1976, pp.573–584, Springer.

Peikert, C., Vaikuntanathan, V. and Waters, B. (2008) 'A framework for efficient and composable oblivious transfer', in Wagner, D. (Ed.): *CRYPTO, Lecture Notes in Computer Science*, Vol. 5157, pp.554–571, Springer.

Pinkas, B., Schneider, T., Smart, N.P. and Williams, S.C. (2009) 'Secure two-party computation is practical', in Matsui, M. (Ed.): *ASIACRYPT, Lecture Notes in Computer Science*, Vol. 5912, pp.250–267, Springer.

Poupard, G. and Stern, J. (1998) 'Generation of shared RSA keys by two parties', in *ASIACRYPT '98: Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security*, pp.11–24, Springer-Verlag.

Rabin, M.O. (1981) 'Fingerprinting by random polynomials', Harvard Aiken Computational Laboratory TR-15-81.

Stern, J.P. (1998) 'A new and efficient all-or-nothing disclosure of secrets protocol', in *Advances in Cryptology ASIACRYPT 98*, pp.357–371, Springer-Verlag.

Wright, R. and Yang, Z. (2004) 'Privacy-preserving Bayesian network structure computation on distributed heterogeneous data', in *Proc. of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.713–718, ACM Press.

Yao, A.C-C. (1986) 'How to generate and exchange secrets (extended abstract)', in *FOCS*, pp.162–167, IEEE.

Notes

[1] The full simulation definition is preferable to the so called 'semi-simulatable security' definition, which only guarantees privacy, but not correctness, in the malicious case. That definition was commonly used for two-party protocols such as oblivious transfer and PIR. It does not enable, however, to use the composition theorem in order to model the resulting building-block protocols as simple calls to a trusted oracle. There are recent efficient constructions of generic protocols which are secure according to this definition (by Lindell and Pinkas, 2007; Jarecki and Shmatikov, 2007), and there are even implementations of the former protocol (Lindell et al., 2008; Pinkas et al., 2009).

[2] For example, the protocol in Indyk and Woodruff (2006) applies the Naor-Nissim (Naor and Nissim, 2001) protocol to a circuit which computes vector operations over the Real numbers and samples from a Bernoulli distribution; in addition it uses symmetric PIR protocols.

[3] In Blake and Kolesnikov (2006), it was shown how to implement a protocol which transfers one of two strings if $w > w'$, and transfers the other string if $w < w'$ (if $w = w'$ the output is random). It is possible to compute the EQ functionality by combining that protocol with a protocol which outputs a specific value if $w = w'$ and a random value otherwise.

[4] After the first step of the protocol the sender has ℓ homomorphic encryptions, one for each letter location, which are each equal to 0 or 1 depending on whether the letters in that location are identical. This is essentially the data that the receiver sends to the sender in many PIR protocols (e.g., Stern, 1998; Chang, 2004; Gentry and Ramzan , 2005 Lipmaa, 2005) where the receiver sends encryptions of the bits of its input. Therefore, it seems natural to use here one of these protocols and perhaps remove one communication round from the protocol. However, these PIR protocols were designed for a setting where the server has a database of size 2^ℓ, whereas the range of possible outputs of our protocol contains only $\ell + 1$ values. As a result our protocol can use a 1-out-of-($\ell + 1$) OT which is more efficient in terms of both computation and communication.

[5] The COTCD protocol is identical to the Jarecki and Shmatikov (JS) (2007) protocol, with an addition of a preliminary step and a verification step. In the preliminary step, both parties receive their auxiliary inputs: the sender receives a value Δ, which is the difference that must hold between its input values, and the receiver receives the committed value of Δ. In the verification step the sender proves to the receiver in zero-knowledge that the committed values, m_0, m_1, have a difference ±Δ. It is important to note that the receiver knows only Com(Δ) and does not learn Δ.

Appendix

A committed oblivious transfer with constant difference

In this Appendix, we describe a protocol, 'COTCD', which is used as a black box in our *bin*HDOT protocol. The protocol performs the following operations:

- Input: The sender $\mathcal{P}_S$ has an auxiliary input Δ, and the receiver $\mathcal{P}_R$ has a commitment of this value, $\texttt{Com}(\Delta)$. In addition, the sender has as input two values (m_0, m_1) satisfying $|m_0 - m_1| = \Delta$, and the receiver has an input $\sigma \in \{0, 1\}$.
- Output: The receiver learns m_σ. In addition, $\mathcal{P}_S$ proves to $\mathcal{P}_R$ that its input (m_0, m_1) has a difference that is equal to ± Δ, namely that $|m_0 - m_1| = \Delta$.

The protocol is similar to committed OT, i.e., to an OT protocol where the parties commit to their inputs, each party sends its commitments to the other party, and each party verifies that the other party's values are equal to the committed values. In addition, the protocol has an additional auxiliary input Δ known to the sender, and a commitment to Δ which is known to the receiver. The protocol ensures that the difference between the two inputs of the sender is equal to ± Δ.

We construct the protocol based on the Jarecki and Shmatikov (2007) committed OT protocol, which is secure against malicious adversaries and is UC-secure in the common reference string model under the DCR assumption (which is also the assumption used to argue about the security of Paillier encryption). The commitment scheme of the protocol is based on the CS encryption scheme (Camenisch and Shoup, 2003), which is, essentially, a semantically secure homomorphic cryptosystem. The homomorphism property is used by our protocol to prove that the difference between the inputs is as required.

We base our construction on the same steps as those of the Jarecki and Shmatikov (JS) (2007) protocol, where we add a preliminary step and a verification step. In the preliminary step, both parties receive their auxiliary inputs: the sender receives a value Δ, which is the difference between its input values, and the receiver receives the committed value of Δ. In the verification step, the sender proves to the receiver that the committed values, (m_0, m_1), have a difference of exactly ± Δ. It is important to note that the receiver knows only $\texttt{Com}(\Delta)$ and does not know Δ.

To sum up, the main changes we make to the JS the protocol are the following:

1 Before starting the protocol, $\mathcal{P}_S$ receives Δ while $\mathcal{P}_R$ receives the commitment of Δ.

2 Before the third step of the protocol, where the receiver learns one of the sender's inputs, the receiver checks whether $| m_0 - m_1 | \overset{?}{=} \Delta$. Since $\mathcal{P}_R$ has the committed values only, $\mathcal{P}_S$ proves to $\mathcal{P}_R$ in zero-knowledge that the committed values satisfy this relation.

We start by introducing the tools and notations of the protocol, describing the ideal functionality implemented by the protocol in Figure 7 and them describe the protocol itself in Figure 8.

A.1 Cryptographic tools and notations

The protocol is based on the same steps as the Jarecki and Shmatikov (JS) protocol, and uses the same cryptographic notation, primitives and tools, which we briefly review here.

A.1.1 Camenish-Shoup (CS) encryption scheme

The CS encryption scheme (Camenisch and Shoup, 2003) is defined as follows.

Common reference string (CRS). A TTP generates a safe RSA modulus $n = pq$, where $p = 2p' + 1$, $q = 2q + 1$, $|p| = |q|$, $p \neq q$ and p, q, p', q' are all primes. In addition it generates random element $g' \in \mathbb{Z}^*_{n^2}$ and an element $g = (g')^2 n$. The common reference string is (n, g), which also defines an element $\alpha = 1 + n$. Also, we treat all multiplications and exponentiations as operation in $\mathbb{Z}^*_{n^2}$.

It is possible to replace the common reference string by a secure computation by the two parties which calculates the values defined (n, g) by the string. This computation, secure against malicious adversaries, can be done using the results of Blackburn et al. (1998) or of Poupard and Stern (1998).

Key generation. The private key is a random triple x_1, x_2, x_3 $\in \left[0, \frac{n^2}{4}\right]$. The public key is $PK = (n, g, \gamma, \hbar, \phi, \mathsf{hk})$ where $\gamma = g^{x_1}$, $\hbar = g^{x_2}$, $\phi = g^{x+3}$ and hk is a key of a colision-resistant keyed hash function $\mathcal{H}$.

Encryption. Consider a plaintext $m \in \left[-\frac{n}{n}, \frac{n}{2}\right]$. A CS encryption of m with PK and label L, which is denoted $\mathsf{CSenc}^L_{PK}(m)$ is a tuple (u, e, v), where $u = g^r$, $e = \alpha^m \gamma^r$ and $v = abs((\hbar\phi^{\mathcal{H}_{\mathsf{hk}}(u,e,L)})^r)$, for $(e / u^{x_1})^2$. $((abs(a) = a$ if $a < \frac{n}{2}$ and $n - a$ if $a \geq \frac{n}{2}$.)

Decryption. For a ciphertext (u, e, v), if $abs(v) = v$ and $u^{2(x_2 + \mathcal{H}_{\mathsf{hk}}(u,e,L)x_3)} = v^2$, then compute $\tilde{m} = (e / u^{x_1})^2$. If n does not divide $\tilde{m} - 1$, then reject; Otherwise compute $\tilde{m}' = \frac{\tilde{m}-1}{n}$ (over the integers), $m' = \tilde{m}' / 2 \bmod n$ and $m = m'$ *rem n*.

A.1.2 Simplified Camenish-Shoup (sCS) (2007) encryption scheme

Jarecki and Shmatikov (2007) proposed a homomorphic, semantically secure variant of CS cryptosystem (Camenisch and Shoup, 2003), which uses a shorter key and allows efficient proofs that a committed plaintext is encrypted under a *committed key*. This variant is denoted sCS.

The group setting (n, g) is the same, and k, k' are parameters that control the quality of the soundness and zero-knowledge properties of proof systems associated with the sCS encryption. Let $k'' = \frac{|n|}{2}$. The sCS scheme requires that $2k + k' < k''$ and $k < p', q'$.

Key generation. The private key is $x \in [0, 2k'']$ and the public key is $y = g^x$.

Encryption. The encryption of m with public key y is $\mathsf{sCSenc}_y(m) = (u, e)$, where $u = g_r$ and $e = \alpha^m y^r$, $r \in_R \left[0, \frac{n}{4}\right]$. The encryption result is in $\left[-\frac{n}{2}, \frac{n}{2}\right]$.

Decryption. The decryption process is the same as in CS decryption, but omitting the CCA checks on v and using x instead of x' in decrypting (u, e).

A.1.3 Commitments

Similar to the JS protocol, we use the CS and sCS encryption scheme as a commitment scheme, where $PK = (n, g, \gamma, \hbar, \phi, \mathsf{hk})$ is a public key which is chosen by a TTP and the security of the commitment scheme requires the CRS model. The commitment on message m is simply its encryption $\mathsf{Com} = \mathsf{Csenc}^L_{PK}(m)$, and the decommitment is the tuple (r, m, L) used to generate this encryption.

A.1.4 Efficient concurrently secure ZK proof systems in the CRS model

We use in our COTCD protocol ZK proofs of knowledge in the CRS model, which are described in the JS paper (2007). The proof systems are three-round honest verifier zero-knowledge (HVZK) proof systems, and are computationally sound and statistical zero-knowledge with a straight line simulator. They can be used together with the compilation technique of Cramer et al. (1994) in order to generate proofs with similar properties for any disjunctive and conjunctive formula of the atomic statements expressible by such proofs. We use the following proof systems:

- $\mathsf{DLEQ} = \{(g, X, \tilde{g}, \tilde{X}) \mid$ there exists x such that $X^2 = g^{2x}$ and $\tilde{X}^2 = \tilde{g}^{2x}\}$. Namely, this is a proof of the equality of the discrete logarithms of X and $\tilde{X}$ to the bases g and $\tilde{g}$, respectively. (This proof is a straightforward adaptation to the setting of group $\mathbb{Z}_{n^2}^*$ of the standard proof for equality of discrete logarithms.)

- $\mathsf{Cot} = \{(i, e', u', e, u, y, C)|$ there exist m, w, s, r such that $C^2 = \alpha^{2m}\gamma^{2w}$, $e'^2 = e^{2e}\alpha^{2m-i*2s}y^{2r}$, and $u'^2 = u^{2s}y^{2r}\}$. In other words, m is committed in the sCS commitment C, and (u', e') is a correct re-encryption of m (performed by the sender in the COTCD protocol), given the (y, u, e) tuple sent by the receiver. [This proof system was described in Jarecki and Shmatikov (2007) where it was denoted Cot, and is an adaptation of the proof systems presented in Camenisch and Shoup (2003).]
- $\mathsf{Com} = \{(\mathsf{Com}, \mathsf{ids})\}$ there exist m, r s.t. $\mathsf{Com} = (u, C, v)$ where $u = g^r$, $C = \alpha^m\gamma^r$, and $v = (\hbar\phi^{\mathcal{H}_{\mathsf{hk}}(u,C,\mathsf{ids})})^r\}$. In other words, Com is a properly formed CS commitment to some message m with label ids. [This proof is a straightforward simplification of the verifiable encryption proof system of the CS scheme in Camenisch and Shoup (2003).]

A.2 The COTCD functionality

The ideal functionality implemented by the COTCD protocol is described in Figure 7. It is similar to the committed OT functionality of JS defined in Jarecki and Shmatikov (2007), but in addition it requires that the difference between the two server inputs is ±Δ. The protocol implementing the COTCD functionality is described in Figure 8 below. It is almost identical to the committed OT protocol of Jarecki and Shmatikov (2007), with the addition of a commitment to Δ, and a verification step which checks that the inputs have a difference of ±Δ. (This step is highlighted in the protocol below.)

The proof of security is similar to that of the committed OT protocol in Jarecki and Shmatikov (2007). The proofs in the verification step can be run in parallel to Step 2 and therefore the number of rounds remains as in the JS protocol.

Jarecki and Shmatikov (2007)presented the crucial aspects of the proof and the idea behind it, we provide a proof of security of the protocol including our changes.

Theorem 5: The protocol securely computes COTCD in the presence of malicious adversaries.

Proof: As in Jarecki and Shmatikov (2007), we prove the security of the protocol using simulator in the hybrid model, assuming that zero-knowledge proofs are performed by a trusted oracle (or party). The simulator acts as an honest party and executes the protocol against malicious parties with random inputs, such that, it simulates the execution in order to learn the input of the other party.

We compare the execution of the protocol between both parties to an execution with a TTP, where TTP executes the functionality introduced in Figure 7.

$\mathcal{P}_S$ is corrupted. The idea of the proof is extracting the input of $\mathcal{P}_S$ by the simulator. The simulator plays as an honest $\mathcal{P}_R$, in addition, it plays a trusted oracle that chooses the CS public key, PK, which is embedded in the CRS and learns both inputs of $\mathcal{P}_S$. Sim chooses PK such it knows SK, CS private key, in order to decrypt the commitments of $\mathcal{A}$. Finally, it sends both inputs to the TTP. Since CS encryption scheme is semantically secure, $\mathcal{P}_S$ cannot learn the input of $\mathcal{P}_R$ or distinguish between real simulation and real execution of the protocol.

More formally, let $\mathcal{A}$ be an adversary controlling $\mathcal{P}_S$, we construct a simulator, Sim, that generates the view of both parties, $\mathcal{A}$ and $\mathcal{P}_R$, in the hybrid model, given only access to $\mathcal{A}$ and to the ideal model.

Also, we assume that Sim

- Plays in the beginning of the protocol a trusted oracle, chooses (SK, PK) of CS encryption scheme and sends PK to $\mathcal{A}$.
- Receives $cid_\Delta = \mathsf{Com}_\Delta$.

1 *Simulation of* Commit, Sim receives from $\mathcal{A}$ the following:

$$\left\langle \mathsf{ComMsg}_{\mathcal{A},0}, \mathsf{ids}_{\mathcal{A}}, \mathsf{Com}_{\mathcal{A}}(m_0)\right\rangle,$$
$$\left\langle \mathsf{ComMsg}_{\mathcal{A},1}, \mathsf{ids}_{\mathcal{A}}, \mathsf{Com}_{\mathcal{A}}(m_1)\right\rangle$$

In addition, Sim chooses randomly $\sigma^{\mathrm{Sim}} \in_R \{0, 1\}$, simulates an honest $\mathcal{P}_R$ and sends $\mathsf{ComMsg}_{\mathrm{Sim}}$, $\mathsf{ids}_{\mathrm{Sim}}$, $\mathsf{Com}_{\mathrm{Sim}}$.

2 *Simulation of* COTCD Step 1, Sim acts as an honest $\mathcal{P}_R$,

- Sets $\mathsf{ids}_{\mathrm{Sim}} = (\mathcal{A}, \mathrm{Sim}, sid, cid_\Delta, cid_{\mathrm{Sim}}, cid_{\mathcal{A},0}, cid_{\mathcal{A},1})$
- Retrieves $\mathsf{Com}_{\mathrm{Sim}} = (u, C, v)$ and its decommitment r.

As in the protocol, it sends to $\mathcal{A}$ the following, $\mathsf{COTCDMsg1}_{\mathrm{Sim}}$, $\mathsf{ids}_{\mathrm{Sim}}$, (u, e, y).

Sim acts as an honest $\mathcal{P}_R$, performs the same steps, where by the end of the step, Sim runs zero-knowledge proof of $\mathsf{ZKDLEQ}(g, u, \gamma, y, C/e) \wedge \mathsf{ZKCom}(PK, \mathsf{Com}_{\mathrm{Sim}}, (\mathrm{Sim}, cid_{\mathrm{Sim}}))$, with $\mathcal{A}$ as the verifier.

3 *Simulation of COTCD Step 2*, as previous steps, Sim acts an honest $\mathcal{P}_R$, retrieves both commitments of m_0, m_1, which committed by $\mathcal{A}$. Sim plays as the verifier in the zero-knowledge proof of $\mathsf{ZKCot}(0, e_0, u_0, e, u, y, C_0) \wedge \mathsf{ZKCot}(1, e_1, u_1, e, u, y, C_1) \wedge \mathsf{ZKCom}(\mathsf{Com}_{\mathcal{A},0}, \mathcal{A}, cid_{\mathcal{A},1})) \wedge \mathsf{ZKCom}(\mathsf{Com}_{\mathcal{A},1}, \mathcal{A}, cid_{\mathcal{A},1}))$.

If Sim does not accept the proofs, it sends ⊥ to the trusted party and halts.

Otherwise, Sim proceeds in the execution of the protocol.

4 *Simulation of verification step.* Sim computes $\mathsf{Com}(m_0 - m_1)$ and $\mathsf{Com}(m_1 - m_0)$, using homomorphic properties of CS scheme. Sim plays as the verifier in the zero-knowledge proof of $\mathsf{ZKCot}(1, e_0/e_1, u_0/u_1, e, u, y, C_\Delta) \vee \mathsf{ZKCot}(1, e_1/e_0, u_1/u_0, e, u, y, C_\Delta)$. If Sim does not accept the proofs, it sends $\perp$ to the trusted party and halts.

Otherwise, if Sim accepts the proofs, extracts m_0, m_1 from the commitments since it knows the private SK of CS encryption scheme.

- If in any step, $\mathcal{A}$ sends $\perp$ or fail in verifying in the zero-knowledge proofs, Sim sends $\perp$ to the trusted party and halts the execution.
- If Sim learns the inputs of $\mathcal{A}$, namely, $\mathcal{A}$ does not cheat, Sim sends to the trusted party m_0, m_1, outputs whatever $\mathcal{A}$ outputs and halts.

After showing the simulation, where Sim learns the input of $\mathcal{A}$ (or $\mathcal{P}_S$), we show that the joint output distribution of $\mathcal{A}$ and $\mathcal{P}_S$ in the hybrid model protocol execution is indistinguishable from the output of Sim and $\mathcal{P}_S$ in the ideal world simulation.

In any step, $\mathcal{A}$ could send $\perp$ to Sim, namely, terminates its execution in the protocol, this could happen in any step of the execution of the protocol and as in hybrid model, where Sim terminates its running, the protocol also terminates and $\mathcal{A}$ does not learn any information.

In *COTCT Step 2 and verification step*, $\mathcal{A}$ has to proof using zero-knowledge to Sim the correctness of its commitments, if Sim does not accept the proofs, it terminates its running, as in the hybrid model execution, the protocol terminates and $\mathcal{A}$ does not learn any information.

$\mathcal{P}_R$ is corrupted. As previous proof, the idea is learning the input of $\mathcal{P}_R$ such that it does not distinguish between simulation and real executions of the protocol. Sim extracts the input of $\mathcal{P}_R$ from the commitment which is provided by $\mathcal{A}$, since it plays as trusted oracle, chooses the public key *PK* such it knows the private key *SK* of CS encryption scheme, learns the appropriate value of $\mathcal{P}_S$ from TTP and sends it to $\mathcal{A}$.

More formally, let $\mathcal{A}$ be an adversary controlling $\mathcal{P}_R$, we construct a simulator, Sim, that generates the view of both parties, $\mathcal{A}$ and $\mathcal{P}_S$, in the hybrid model, given only access to $\mathcal{A}$ and to the ideal model.

Also, in this simulation, we assume that Sim:

- Knows Δ
- Plays in the beginning of the protocol a trusted oracle, chooses (*SK*, *PK*) of CS encryption scheme and sends *PK* to $\mathcal{A}$.

1 *Simulation of* Commit, Sim receives from $\mathcal{A}\langle \mathsf{ComMsg}_{\mathcal{A}}, \mathsf{ids}_{\mathcal{A}}, \mathsf{Com}_{\mathcal{A}}(\sigma)\rangle$. Sim extracts $\sigma_{\mathcal{A}}$, the input of $\mathcal{A}$, since it knows *SK* and can decrypt the commitment of $\mathcal{A}$, sends $\sigma_{\mathcal{A}}$ to the trusted party and leanrs $m = m_{\sigma_{\mathcal{A}}}$.

2 Sim continues in the *simulation of* Commit, chooses two messages m_0^{Sim}, m_1^{Sim}, such that $m_{\sigma_{\mathcal{A}}}^{\mathrm{Sim}} = m$ and $m_{1-\sigma_{\mathcal{A}}}^{\mathrm{Sim}} = m + \Delta$ and sends $\mathcal{A}$ two commitments of the messages as in Commit step.

3 *Simulation of* COTCD Step 1, Sim receives $\langle \mathsf{COTCDMsg1}_{\mathcal{A}}, \mathsf{ids}_{\mathcal{A}}, (u, e, y)_{\mathcal{A}}\rangle$ from $\mathcal{A}$ and plays the verifier in the zero-knowledge proof of $\mathsf{ZKDLEQ}(g, u, \gamma, y, C/e) \wedge \mathsf{ZKCom}(PK, \mathsf{Com}_{\mathcal{A}}, (\mathcal{A}, cid_{\mathcal{A}}))$.

If Sim did not accept the proof, it terminates the execution of the protocol and send $\perp$ to the trusted party. Otherwise, proceeds in the protocol.

- As previous simulation, if in any step, $\mathcal{A}$ sends $\perp$ or fail in verifying in the zero-knowledge proofs, Sim sends $\perp$ to the trusted party and halts the execution.
- If Sim learns the inputs of $\mathcal{A}$, it outputs whatever $\mathcal{A}$ outputs and halts.

As previous, we show that the joint output distribution of $\mathcal{A}$ and $\mathcal{P}_S$ in the hybrid model protocol execution is indistinguishable from the output of Sim and $\mathcal{P}_R$ in the ideal world simulation.

In any step, $\mathcal{A}$ could send $\perp$ to Sim, namely, terminates its execution in the protocol, this could happen in any step of the execution of the protocol and as in hybrid model, where Sim terminates its running, the protocol also terminates and $\mathcal{A}$ does not learn any information.

In COTCT Step 1, $\mathcal{A}$ has to proof using zero-knowledge to Sim the correctness of its commitment, if Sim does not accept the proofs, it terminates its running, as in the hybrid model execution, the protocol terminates and $\mathcal{A}$ does not learn any information. □

Figure 6 Run time of $\text{Similarity}_{\tau}^{k:n}$ and *bin*HDOT

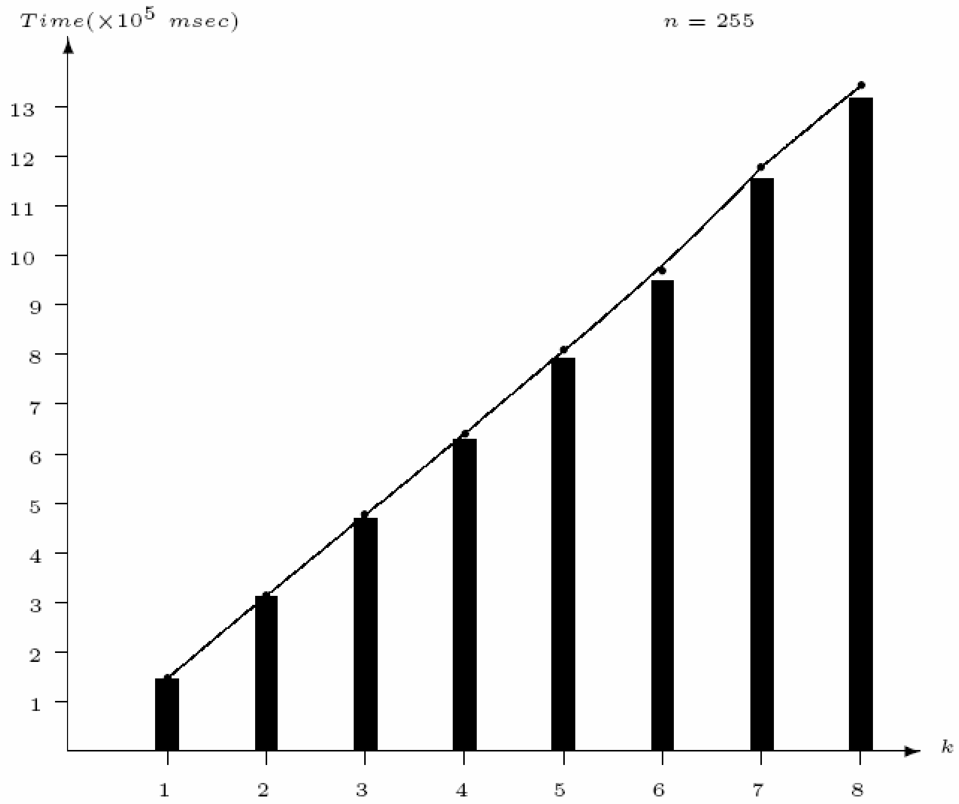


Figure 7 *COTCD* ideal functionality

Input: $\mathcal{P}_R$ and $\mathcal{P}_S$ receives their inputs to the $\mathcal{F}_{COTCD}$, $\mathcal{P}_R$ receives $\sigma \in \{0, 1\}$ and $\mathcal{P}_S$ receives m_0, m_1.

Additionally, they receive auxiliary inputs, $\mathcal{P}_R$ receives $\mathsf{Com}(\Delta)$ and $\mathcal{P}_S$ receives Δ and checks that $|m_0 - m_1| = \Delta$.

Commit: Upon receiving a $\langle \mathsf{ComMsg}, (\mathcal{P}_i, cis), m\rangle$ message from $\mathcal{P}_i$, $\mathcal{F}_{COTCD}$ records the $((\mathcal{P}_i, cid), m)$ pair and broadcasts $\langle \mathsf{Committed}, (\mathcal{P}_i, cid), m\rangle$. Here m can be either a message in the prescribe message space or a special symbol $\perp$.

StartCOT: Upon receiving $msg = \langle \mathsf{StartCOTCD}, (\mathcal{P}_S, \mathcal{P}_R, sid, cid_R, cid_\Delta, cid_{S,0}, cid_{S,1})\rangle$ from $\mathcal{P}_S$, $\mathcal{F}_{COTCD}$ verifies that it has records $((\mathcal{P}_R, cid_R), m_R)$, $((\mathcal{P}_S, cid_{S,0}), m_{S,0})$, $((\mathcal{P}_S, cid_{S,1}), m_{S,1})$, $((\mathcal{P}_S, cid_\Delta), \Delta)$. It, also, checks that $|m_0 - m_1| = \Delta$ and that, $m \neq \perp$. If this fails, $\mathcal{F}_{COTCD}$ ignores this message; Otherwise, $\mathcal{F}_{COTCD}$ records *msg* and forward it to $\mathcal{P}_S$.

CompleteCOTCD: Upon receiving $\langle \mathsf{CompleteCOTCD}, (\mathcal{P}_S, \mathcal{P}_R, sid, cid_R, cid_\Delta, cid_{S,0}, cid_{S,1})\rangle$ from $\mathcal{P}_S$, $\mathcal{F}_{COTCD}$ verifies that it has a record $\langle \mathsf{StartCOTCD}, \mathsf{ids}\rangle$, where $\mathsf{ids} = (\mathcal{P}_S, \mathcal{P}_R, sid, cid_R, cid_\Delta, cid_{S,0}, cid_{S,1})$. $\mathcal{F}_{COTCD}$ looks up records $((\mathcal{P}_S, cid_{S,0}), m_{S,0})$, $((\mathcal{P}_S, cid_{S,1}), m_{S,1})$ and $((\mathcal{P}_S, cid_\Delta), \Delta)$, and checks:

- $m_{S,0} \neq \perp$
- $m_{S,1} \neq \perp$
- $|m_{S,0} - m_{S,1}| = \Delta$, a verification step to the difference between the messages.

If anything fails, $\mathcal{F}_{COTCD}$ ignores this message.

Otherwise $\mathcal{F}_{COTCD}$ looks up the record $(\mathcal{P}_R, cid_R), m_R)$. If $m \notin \{0, 1\}$, $\mathcal{F}_{COTCD}$ sends a special message $\langle \mathsf{COTCDFailed}, \mathcal{P}_S, \mathcal{P}_R, sid\rangle$ to $\mathcal{P}_R$. Otherwise $\mathcal{F}_{COTCD}$ sends $\langle \mathsf{CompleteCOTCD}, \mathsf{ids}, (m_{S,b}, b)\rangle$ to $\mathcal{P}_R$ for $b = m =$.

Figure 8 Protocol COTCD, for committed OT with constant difference

Common reference string: A committed instance of the public key of t he CS encryption scheme $PK = (n, g, \gamma, \hbar, \phi, \mathsf{hk})$.

Auxiliary input: $\mathcal{P}_S$ receives Δ and $\mathcal{P}_R$ receives a commitment Com_Δ which is indexed by an identifier cid_Δ (in our application the parties will invoke this protocol many times, and use the same Δ value and commitment in all these invocations).

Input: $\mathcal{P}_S$'s input contains two messages (m_0, m_1), where $|m_0 - m_1| = \Delta$, and $m_0, m_1, \Delta \in \left[0, \frac{n}{2}\right]$. $\mathcal{P}_R$'s input is $\sigma \in \{0, 1\}$.

Output: $\mathcal{P}_R$ learns m_σ while $\mathcal{P}_S$ does not learn any information.

Commit: For player $\mathcal{P}_i$, on commitment instance *cid* and message *m*: Player $\mathcal{P}_i$ sets $\mathsf{ids} = (\mathcal{P}_i, cid)$, $\mathsf{Com} = \mathsf{CSenc}_{Pk}^{\mathsf{ids}}(m)$, and broadcasts $\langle \mathsf{ComMsg}, \mathsf{ids}, \mathsf{Com} \rangle$.

Protocol execution: Receiver $\mathcal{P}_R$ executes a COTCD instance *sid* with sender $\mathcal{P}_S$. $\mathcal{P}_R$'s bit σ is committed in Com_R, $\mathcal{P}_S$'s messages m_0, m_1 are committed in $\mathsf{Com}_{S,0}$, $\mathsf{Com}_{S,1}$. Let cid_R, $cid_{S,0}$, $cid_{S,1}$ be the identifiers for these commitments.

COTCD Step 1: $\mathcal{P}_R$ sets $\mathsf{ids} = (\mathcal{P}_S, \mathcal{P}_R, sid, cid_\Delta, cid_R, cid_{S,0}, cid_{S,1})$, retrieves $\mathsf{Com}_R = (\tilde{u}, C, \tilde{v})$ and its decommitment $r \in \left[0, \frac{n}{4}\right]$. Note that $C = \alpha^\sigma \gamma^r$. $\mathcal{P}_R$ picks $x \in \left[0, \frac{n}{4}\right]$, and computes

$$y = g^x, u = g^r, e = \alpha^\sigma y^r$$

$\mathcal{P}_R$ sends $\langle \mathsf{COTCDMsg1}, \mathsf{ids}, (u, e, y) \rangle$ to $\mathcal{P}_S$, and runs the proof system $\mathsf{ZKDLEQ}(g, u, \gamma, y, C/e) \wedge \mathsf{ZKCom}(PK, \mathsf{Com}_R, (\mathcal{P}_R, cid_R))$ with $\mathcal{P}_S$, where it operates as the prover.

COTCD Step 2: After receiving $\langle \mathsf{COTCDMsg1}, \mathsf{ids}, (u, e, y) \rangle$ which was sent to $\mathcal{P}_S$ from $\mathcal{P}_R$, $\mathcal{P}_S$ retrieves messages m_0, m_1 committed in $\mathsf{Com}_{S_0} = (\tilde{u}_0, C_0, \tilde{v}_0)$ and $\mathsf{Com}_{S_1} = (\tilde{u}_1, C_1, \tilde{v}_1)$. Note that $C_i = \sigma_{m_i} \gamma_{r_{mi}}$ for some r_{m_i}. $\mathcal{P}_S$ creates two 'COTCD-encryptions' for $i = 0; 1$:

$$e_i = e^{s_i} \alpha^{m_i - i * s_i} y^{r_i} \text{ and } u_i = u^{s_i} g^{r_i}$$

using random even values $s_i \in [0, 2n]$ and $r_i \in \left[0, \frac{n}{2}\right]$. If $\mathcal{P}_R$ passed its proof in Step 1, $\mathcal{P}_s$ sends message $\langle \mathsf{COTCDMsg2}, \mathsf{ids}, (u_0, e_0, u_1, e_1) \rangle$ to $\mathcal{P}_R$, and performs, with $\mathcal{P}_R$ as the verifier, a proof that $\mathsf{ZKCot}(0, e_0, u_0, e, u, y, C_0) \wedge \mathsf{ZKCot}(1, e_1, u_1, e, u, y, C_1) \wedge \mathsf{ZKCom}(\mathsf{Com}_{S,0}, (P_S, cid_{S_0})) \wedge \mathsf{ZKCom}(\mathsf{Com}_{S,1}, (P_S, cid_{S_1}))$.

Verification step: In addition, the parties run the following step to verify that the difference between m_0 and m_1 is $\pm\Delta$. This is the main part in which the protocol is different from the protocol in Jarecki and Shmatikov (2007).

$\mathcal{P}_R$ computes two commitments, $\mathsf{Com}(m_0 - m_1)$ and $\mathsf{Com}(m_1 - m_0)$, by using the homomorphic properties of the CS scheme and computing $\mathsf{Com}(m_0)/\mathsf{Com}(m_1)$ and $\mathsf{Com}(m_1)/\mathsf{Com}(m_0)$ (namely, $\mathcal{P}_R$ computes $(e_0/e_1, u_0/u_1)$ and $(e_1/e_0, u_1/u_0)$).

$\mathcal{P}_S$ performs, with $\mathcal{P}_R$ as the verifier, a proof that one of these two commitments is a commitment to Δ. Namely, $\mathcal{P}_S$ proves that $\mathsf{ZKCot}(1, e_0/e_1, u_0/u_1, e, u, y, C_\Delta) \vee \mathsf{ZKCot}(1, e_1/e_0, u_1/u_0, e, u, y, C_\Delta)$.

If $\mathcal{P}_S$ passes its verification of the zero-knowledge proofs both parties continue to Step 3; Otherwise, $\mathcal{P}_R$ rejects.

COTCD Step 3: $\mathcal{P}_R$ decrypts the sCS ciphertext (u_σ, e_σ) and obtains m_σ. If $\mathcal{P}_S$ passed its proof in Step 2, then $\mathcal{P}_R$ outputs m_σ; Otherwise $\mathcal{P}_R$ rejects.

Comment: We have considered simplifying the steps of the protocol by removing the proof that the commitments to m_0 and m_1 are properly formed (namely the proof in Step 2 that $\mathsf{ZKCom}(\mathsf{Com}_{S,0}, (P_S, cid_{S_0})) \wedge \mathsf{ZKCom}(\mathsf{Com}_{S,1}, (P_S, cid_{S_1}))$. After all, we are not interested in the sender committing to these values but rather in ensuring that the difference of these values is $\pm\Delta$. We cannot do that, however, since removing these proofs might enable the sender to commit to two random values that have a difference of Δ but are otherwise unknown to the sender.