Breaking and Fixing Secure Similarity Approximations: Dealing with Adversarially Perturbed Inputs

Evgenios M. Kornaropoulos
Dept. of Computer Science, Brown University
evgenios@cs.brown.edu

Petros Efstathopoulos
Symantec Research Labs
petros_efstathopoulos@symantec.com

***Abstract*—Computing similarity between data is a fundamental problem in information retrieval and data mining. To address the relevant performance and scalability challenges, approximation methods are employed for large-scale similarity computation. A common characteristic among all privacy-preserving approximation protocols based on *sketching* is that the sketching is performed *locally* and is based on *common randomness*.**

In the semi-honest model the input to the sketching algorithm is independent of the common randomness. We, however, consider a new threat model where a party is allowed to use the common randomness to perturb her input 1) offline, and 2) before the execution of any secure protocol so as to steer the approximation result to a maliciously chosen output. We formally define *perturbation attacks* under this adversarial model and propose two attacks on the well-studied techniques of minhash and cosine sketching. We demonstrate the power of perturbation attacks by measuring their success on synthetic and real data.

To mitigate such perturbation attacks we propose a *server-aided* architecture, where an additional party, the server, assists in the secure similarity approximation by handling the common randomness as private data. We revise and introduce the necessary secure protocols so as to apply minhash and cosine sketching techniques in the server-aided architecture. Our implementation demonstrates that this new design can mitigate offline perturbation attacks without sacrificing the efficiency and scalability of the reconstruction protocol.

1. Introduction

Analyzing data to extract important insights particular system has become a tool of central importance to most lines of business and science. Many of the tasks data analysis aims to achieve can be reduced to the problem of detecting similarity between data points in sets. In scenarios such as legal discovery, forensics, and genomic analytics, it is important to detect similarity between high dimensional data even in the presence of *strict privacy requirements*. As computing exact similarity metrics on very large datasets is prohibitively expensive, state-of-the-art methods *approximate* the similarity function that needs to be computed by working with a succinct representation of the data that is called a *sketch*.

Sketching is the mainstream approach for efficiently approximating a plethora of functions [11], [12], [16], [18], [25], [36], [42], [47], [57], [58]. The seminal work by Feigenbaum *et al.* [27] set the foundation for secure multiparty computation of approximation functions. Since then, researchers proposed a variety of secure approximations *based on sketching* for several application areas. Melis *et al.* [45] propose scalable and secure protocols to approximate statistics on large data streams. Blundo *et al.* [9] propose secure approximation algorithms for iris matching, multimedia content similarity, and document similarity. The community has made several important steps towards private computation on genomic data in a time-efficient and scalable manner [3], [17], [22], [50]. Wang *et al.* [62] demonstrate the power of carefully-engineered secure approximation by running a privacy-preserving similarity query for a human genome on 1 million records distributed across the U.S., in a couple of minutes. In their CCS'15 work they point out:

> "Generally a streaming algorithm consists of phases to locally compute the sketches and those to jointly combine the sketches. Feigenbaum *et al.* [27] pointed out that it is unclear how to prove it secure if a secure computation protocol is not used to compute the sketches, (although no actual attacks are identified)."

In this work we capture the capabilities of the adversary in *a new threat model, and design the first attack of this kind*. The sketching protocol in [27] has two phases: 1) the sketching function is applied locally by each party, and 2) the reconstruction function is performed via secure multiparty computation. Our offline attack is mounted on the first phase by a data owner and exploits the fact that 1) the *randomness* of the sketching algorithm is known to all the participants, and 2) the sketch algorithm is performed *locally*. In this threat model, the *only action* the attacker is allowed to take is to change the input data to the sketching algorithm; that is, the computation of the sketch. The computation of the reconstruction, the communication, as well as the input to the reconstruction, remain untouched. As a result, *the correctness of the approximation* is violated since the adversary can steer any future approximation between

the perturbed data and any other data point to an incorrect output, *regardless of the secure computation protocols of the second phase*.

There are several scenarios where a data owner has the incentive to mount a perturbation attack. In the case of legal forensic discovery, which is a $ 9.9 billion market [23], the plaintiff company is interested in correctly approximating similarity between documents so as to discover important evidence that can strengthen their case, while the defending company might prefer to masquerade evidence by causing mis-approximation. In the case of customized medical treatment a patient uses her genomic data to discover patients with similar gene-expression that are already treated for the same disease; thus discovering similarities is to the benefit of the user. In a case, however, where a DNA sample is found in a crime scene, a suspect benefits from mis-approximating the similarities with the found sample. Thus the intention of the participant depends on the stakes of the outcome of the approximation function.

To mitigate perturbation attacks we propose a new *server-aided secure approximation architecture* that requires the participation of three parties, as opposed to two of the previous designs. A new honest-but-curious entity—the server—stores the common randomness which is treated as private information. A user runs a protocol with the server to build an encrypted sketch, as opposed to the local computation of the previous model's first phase. During this protocol the user doesn't learn any information about the common randomness and the server doesn't learn any information about the user's data. We emphasize that the sketch-generation takes place *only once for each data point*, and the sketch can be reused for any future pairwise approximation. Under this new server-aided framework the users *do not have direct access* to the common random input (only via communication with the server) and thus they can not mount an offline perturbation attack. In this paper, we devise and implement new secure protocols in order to securely generate minhash and cosine sketches in our proposed architecture. Given a pair of sketches[1] our implementation achieves throughput of 30-600 approximations per second for data points with hundreds of dimensions.

Related Work. Aside from the secure sketching protocols mentioned earlier, there is a rich body of protocol that devise a combination of semi-homomorphic cryptosystems and garbled circuits to operate on encrypted data [4], [10], [40], [63].

The work by Mironov *et al.* [46] introduces the model of *sketching in adversarial environments* which is different in certain ways from what we consider in our work. Specifically, the work in [46] studies a model where a *single* party adversarially chooses the input *for all other parties* while they approximate joint functions on the adversarially chosen input. In their model, the adversarial inputs are provided to the parties in an *on-line* manner and thus the users update the sketch incrementally without being able to store the original information, much like in one-pass streaming algorithms. In our work, each party uses her own data which is stored locally. Our model is different from the data stream model, and follows more closely the published work on privacy-preserving sketches discussed above.

The work by Naor *et al.* [49] introduces a new *adversarial model for Bloom filters*. A Bloom filter is used for set-membership testing; it might output a false-positive response due to the probabilistic design that prioritizes space and time efficiency over accuracy. The adversary in this threat model issues a series of adaptively chosen queries and his goal is to predict which non-queried element will result into a false-positive. The threat model of [49] is somewhat similar to our model, in the sense that both adversaries exploit the used randomness so as to violate the correctness of the computation. In terms of differences, our adversary has direct access to the randomness used, whereas for the case of [49] the adversary has only oracle access via the responses of the Bloom filter. Furthermore in our work sketching is just the first phase of the computation and the second phase consists of a secure computation protocol; on the contrary the work of [49] does not involve any form of encryption or secure computation.

There's a significant body of research focusing on the attack vectors that lay in the intersection of machine learning and privacy-preserving mechanisms [5], [15], [19], [28], [29], [44]. The line of research closer to our proposed attack is the work on Deep Learning in adversarial settings. Some papers [53], [59] show how an adversary can *craft her input* so as to maximize the prediction error of a deep neural network (DNN), while others propose mitigation techniques [54]. In a realistic scenario of a DNN under attack the adversary can only observe the output of a chosen input, since the proprietary DNN is stored at the company that provides the classification service. On the contrary, our attacks are based on a more realistic assumption (the attacker has local access to the common randomness by design [27]), can be mounted offline (impossible to mitigate with rate-limiting techniques), and can be generalized to a broader family of approximation functions than the attacks on DNNs.

Our Contributions. Our work makes the following contributions:

- We identify and formalize the notion of *perturbation attacks* for secure multiparty approximation protocols. We demonstrate their power by proposing two such perturbation attacks. The first attack uses probabilistic arguments, and is mounted on the *minhash sketching* technique, which is deployed to measure the Jaccard similarity between two sets. The second attack formulates a non-convex high-dimensional constrained optimization problem, and is mounted on the *cosine sketching* technique, which is deployed to measure the cosine similarity between two vectors. We apply the proposed attacks on both real and synthetic data.
- We propose a new *server-aided* approach that miti-

1. The parameterization, and consequently the efficiency, of the sketching instantiation depends on the approximation guarantees.

gates offline perturbation attacks. In our new setup, a server has exclusive access to the common randomness, and is assisting the clients in the sketch computation. In the new design, a user does not learn any information about the common random input. Additionally, the server doesn't learn any information about the user's data.
- We devise new secure protocols for the server-aided design for the case of *minhash* sketching and *cosine* sketching. Furthermore, we implement the protocols and evaluate their performance.

2. Preliminaries and Background

2.1. k-Independent Hashing

Space and time-efficient hash functions provide rigorous guarantees about the distribution of their values, such a family is the family of k-*independent* hash functions. A well studied construction of a k-independent family is based on polynomials of degree $k-1$. Let $p > |U|$ be a prime and $a_0, a_1, \ldots, a_{k-1} \in \mathbb{Z}_p$ be independent and uniformly chosen values over the prime field, then we have:

$$h(x) = (\alpha_{k-1}x^{k-1} + \ldots + \alpha_1 x + \alpha_0) \mod p.$$

For $p > m$, the above family of hash functions is statistically close to the k-independence property.

2.2. Secure Sketching

Exact similarity computation between two data points takes at least linear time with respect to the size of the data, since we need to parse the data item for *every comparison* regardless of the similarity function. A way to overcome this overhead is to settle with an *approximation* of similarity as opposed to exact computation.

Definition 2.1. *(Def. 10.1 in [48]) A randomized algorithm gives an* (ϵ, δ)-approximation *for the value* ν *if the output* ν' *of the algorithm satisfies,* $\Pr(|\nu' - \nu| \leq \epsilon\nu) \geq 1 - \delta$.

We are interested in *sketching techniques* that are well-studied and widely applied in the area of data-mining and information retrieval [11], [12], [16], [18], [25], [36], [42], [47], [57], [58]. A benefit of sketching is that the succinct summary of the data, i.e., *the sketch*, is built once and can be reused in future pairwise approximations. Thus the super-linear overhead occurs only during the construction of the sketch which significantly speeds up the total time performance over a series of similarity approximations. The notion of a sketching protocol is defined as:

Definition 2.2. *(Def. 8 in [27]) A* sketching protocol *for a 2-argument function* $f : \{0,1\}^* \times \{0,1\}^* \to \mathbb{N}$ *is defined by:*

- *A* sketching function*,* $\mathcal{S} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ *mapping one input and a random string to a* sketch *consisting of a (typically) short string.*
- *A (deterministic)* reconstruction function $\mathcal{G} : \{0,1\}^* \times \{0,1\}^* \to \mathbb{R}$*, mapping a pair of sketches to an approximate output.*

On inputs $\alpha, \beta \in \{0,1\}^n$*, the protocol proceeds as follows. First, Alice and Bob locally compute a sketch* $\sigma_A = \mathcal{S}(\alpha, r_{cmn})$ *and* $\sigma_B = \mathcal{S}(\beta, r_{cmn})$ *respectively, where* r_{cmn} *is a common random input. Then, the parties exchange sketches, and both output locally* $\hat{f} = \mathcal{G}(\sigma_A, \sigma_B)$*. We denote by* $\hat{f}(\alpha, \beta)$ *the randomized function defined as the output of the protocol on inputs* α, β*. A sketching protocol as above is said to* (ϵ, δ)*-approximate* f *if* $\hat{f}$ (ϵ, δ)*-approximates* f*.*

Following the terminology of Goldreich for multiparty computation (Section 7.2 [32]) we capture the above process with the following *functionality*:

$$\mathcal{F}_{\mathsf{Approx}}\big((\alpha, r_{cmn}), (\beta, r_{cmn})\big) \to (\hat{f}(\alpha, \beta), \hat{f}(\alpha, \beta)) \quad (1)$$

, where the first (resp. second) pair is the input of client C_A (resp. client C_B) and the output to both parties is the (ϵ, δ)-approximation $\hat{f}(\alpha, \beta)$. We note here that if the clients execute the sketching computation with *different randomness* then the output of the reconstruction is meaningless [2], thus the randomness must be the same. We emphasize that α, β are user-provided inputs and their legitimacy relies on the honesty and intention of the user.

A metric space is a set X accompanied with a *distance function* $d : X \times X \to \mathbb{R}$, or simply *distance*, that measures the distance between points $x, y \in X$. We are interested in the approximation of distance functions from which we can derive the similarity. The terms "distance" and "similarity" are used interchangeably in the rest of the work, in the next subsection we show how the two concepts relate for the distance measures that we studying.

2.3. Similarity Approximation

Approximating Jaccard Similarity. The *Jaccard similarity coefficient* (or Jaccard index) measures the similarity between two sets. Formally, given a pair of sets $S_1, S_2 \subseteq U$ the Jaccard similarity coefficient and the Jaccard distance d_{Jac} are defined as:

$$J(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}, d_{Jac}(S_1, S_2) = 1 - J(S_1, S_2).$$

Minwise hashing [11], [12], or *minhashing*, is a technique for approximating the Jaccard index that has been successfully applied to numerous problems (e.g., [12], [42], [47], [57], [60]). Even though the analysis of the approximation is based on random permutations [11], in practice we use minhash functions that are defined as $h_i^{min}(S) = min_{x \in S}(h_i(x))$, where h_i is a k-independent hash function. Using κ distinct minhash functions one can build a *minhash sketch*, also called minhash signature, $\sigma(S)$ for input set S.

2. This is equivalent to using different hash functions for the approximation of Jaccard similarity, or using different random vectors for the approximation of the cosine similarity.

Given two minhash sketches we approximate the Jaccard distance $\hat{d}_{Jacc}$ as follows:

$$\hat{d}_{Jacc}(S_1, S_2) = \frac{1}{\kappa} d_H(\sigma(S_1), \sigma(S_2)), \quad \sigma(S) = (h_1^{min}(S), \ldots, h_\kappa^{min}(S)), \tag{2}$$

where d_H denotes the hamming distance between the two input arguments. The common random input r_{cmn} from Definition 2.2 is used to initialize minhash functions.

Mitzenmacher *et al.* [47] introduced an approximation technique using odd sketches. An *odd sketch* of set S, denoted as $odd(S)$, consists of (A) a bit array T of size u and (B) a hash function $h_{odd} : U \to [0, u-1]$. In order to approximate the Jaccard similarity via odd sketches one uses the values of the minhash sketch $\sigma(S) = (x_1,\ldots,x_\kappa)$ as the input set for the odd sketch. Whenever an item $x_i = h_i^{min}(S)$, where $i \in [1, \kappa]$, is hashed to the odd sketch T using function h_{odd}, the bit in position $h_{odd}(x_i)$ of T is flipped. We approximate the Jaccard index as follows [47]:

$$\hat{J}_{odd}(S_1, S_2) = 1 + \frac{u}{4\kappa} \ln\left(1 - \frac{2|odd(\sigma(S_1)) \Delta odd(\sigma(S_2))|}{u}\right) \tag{3}$$

, where $|odd(\sigma(S_1)) \Delta odd(\sigma(S_2))|$ denotes the number of 1s in the sketch resulted after the exclusive-or operation over the odd sketches, κ denotes the number of independent minhash values, and u denotes the size of the odd sketch. Jaccard distance is approximated using eq. (3), as $\hat{d}_{Jacc}(S_1, S_2) = 1 - \hat{J}_{odd}(S_1, S_2)$. The common random input r_{cmn} is used to initialize h_{odd} and $h_1^{min}, \ldots, h_\kappa^{min}$. Thus all the parties of the sketching protocol (see Definition 2.2) generate the same hash functions.

Approximating Cosine Similarity. The work of Charikar [16] introduced the notion of *cosine sketching* commonly used [25] to estimate the similarity between two vectors. Formally, let $\vec{v_1}, \vec{v_2} \in \mathbb{R}^n$ the *cosine similarity* as

$$C(\vec{v_1}, \vec{v_2}) = \frac{\vec{v_1} \cdot \vec{v_2}}{\|\vec{v_1}\|_2 \|\vec{v_2}\|_2}, d_{cos}(\vec{v_1}, \vec{v_2}) = 1 - C(\vec{v_1}, \vec{v_2})/2, \tag{4}$$

where $\| \cdot \|_2$ is the Euclidean norm of the vector. The resulting similarity $C(\vec{v_1}, \vec{v_2})$ ranges from -1 to 1 which is interpreted as completely opposite and as exactly the same, respectively. The cosine sketching technique is based on sign random projections. Let $\vec{v} \in \mathbb{R}^n$ be a unit vector[3], then the cosine sketch is a κ-dimensional bit vector $\sigma(\vec{v}) = (\sigma_1, \ldots, \sigma_\kappa)$. The components σ_i for $i \in [1, \kappa]$ and the symmetric cosine sketch distance [43] are defined as:

$$\sigma_i = \begin{cases} 1, & \text{if } \vec{w_i}^T \cdot \vec{v} \geq 0 \\ 0, & \text{if } \vec{w_i}^T \cdot \vec{v} < 0, \end{cases}, \hat{d}_{cos}(\vec{v_1}, \vec{v_2}) = \frac{d_H(\sigma(\vec{v_1}), \sigma(\vec{v_2}))}{\kappa}, \tag{5}$$

where $\vec{w_i} \in \mathbb{R}^n$ is sampled uniformly at random from the n-dimensional unit hypersphere. The common random input r_{cmn} is used to initialize the vectors $\vec{w_i}$, for $i \in [1, \kappa]$.

3. In case the input vector is not unit we convert it by normalizing.

2.4. Semi-Homomorphic Cryptosystems

We use the described notation to highlight that messages are encrypted under different cryptosystems.

Paillier Cryptosystem. The Paillier cryptosystem [52] is semantically secure. With the term $[m]$ we denote the encryption of message m under the key pair $K_P = (PK_P, SK_P)$; from the additive homomorphism we have that $[m_1] \cdot [m_2] = [m_1 + m_2]$.

Goldwasser-Micali Cryptosystem. The Goldwasser - Micali (GM) cryptosystem [33] is semantically secure. With the term $|m|$ we denote the encryption of the bit m under the key pair $K_{GM} = (PK_{GM}, SK_{GM})$; from the GM homomorphism we have that $|m_1| \cdot |m_2| = |m_1 \oplus m_2|$, where $\oplus$ is the XOR operation.

Damgård-Geisler-Krøigaard Cryptosystem. The Damgård-Geisler-Krøigaard (DGK) cryptosystem [20], [21] is semantically secure. The DGK cryptosystem is considered to be much more efficient [8], [26], [41] than Paillier due to its small plain-text space. With the term $\langle m \rangle$ we denote the encryption of message $m \in \mathbb{Z}_u$ under the key pair $K_{DGK} = (PK_{DGK}, SK_{DGK})$. Similarly to Paillier, DGK is additively homomorphic; moreover, it embeds reductions modulo u to its homomorphic operations, therefore $\langle m_1 \rangle \cdot \langle m_2 \rangle = \langle (m_1 + m_2) \mod u \rangle$.

3. Threat Model

In this work we consider a new threat model where the adversary can maliciously perturb *only* her input to the sketching algorithm which is executed offline and locally. This new adversary does not interfere with the computation of the sketching, the computation of the reconstruction, and the communication, i.e., she acts in a semi-honest fashion after the perturbation of the input data. Thus, our threat model is *not* the honest-but-curious, it is a very restricted version of the malicious model where the proposed adversary can only alter her data offline, by using the already distributed common randomness. As we show in later sections, even though our threat model does not differ much from the semi-honest, it allows for devastating attacks on the correctness of the approximation. This adversarial behavior is challenging to detect since the perturbation takes place offline, on her own data. Other works in the area of MPC consider adversaries that attempt to learn to input of another party, this is not the case here, the goal of the adversary is to masquerade her own data with respect to the approximation mechanism.

Consider the following class of protocols that compute the functionality $\mathcal{F}_{\text{Approx}}$ as it is described in Section 2.2.

Class of Protocols for $\mathcal{F}_{\text{Approx}}$

- **Step 1:** Generate and distribute the common random input r_{cmn} to all the parties.
- **Step 2:** Each party inputs her data and r_{cmn} so as to locally compute the sketching function $\mathcal{S}$.
- **Step 3:** Parties run an MPC protocol that outputs the result of the reconstruction function $\mathcal{G}$.

Attack Surface of $\mathcal{F}_{\text{Approx}}$. We assume that the adversary participates in the above protocol. We distinguish two possible *offline* attacks on this class of protocols, the attacker can: 1) deviate from the correct execution of the locally computed sketching, and/or 2) execute the sketching correctly, but corrupt its output—and therefore the input to the reconstruction function $\mathcal{G}$. Both attacks can be detected using verifiable computation [30], [55], i.e., provide proof of correctness for the computation and the output of $\mathcal{S}$. Addressing such mitigations is outside the scope of our work and is left as future work. We focus on the remaining attack surface: since cryptographic techniques exist to detect the above attacks, the last resort for the adversary is to perturb the input to the sketching function.

Perturbing the input to $\mathcal{S}$. To capture the remaining attack surface, in the new threat model we extend the above class of protocols by allowing the adversary to locally execute a function right before Step 2. Specifically, the adversary executes a randomized function Perturb that takes as an input the data point α and the common random input r_{cmn} outputted by Step 1. Function Perturb runs locally, without any interaction, and outputs a value α^+ that will serve as the new input to the sketching function $\mathcal{S}$. We emphasize that after the execution of Perturb the adversary *behaves in a semi-honest fashion*, i.e., she honestly follows the sketching function and honestly executes the MPC protocol. Thus, in our new threat model the only malicious activity of the adversary is the local execution of Perturb.

Other work has considered a similar adversarial behavior. Raghunathan *et al.* [56] consider adversaries that adaptively choose the plaintext distribution after seeing the public key of a Deterministic Public-Key Encryption scheme. Abadi *et al.* [2] propose adversaries that adaptively choose the distribution of plaintexts after seeing the parameters of a Message Locked Encryption scheme.

4. Perturbation Attack

In this Section we define *perturbation attacks* on the class of protocols defined in Section 3. A successful attack on secure sketching protocols for a two-argument *distance function* yields a perturbed input such that althought the pair (original input, perturbed input) is really close with respect to the corresponding distance function, the approximation instantiation of the above pair appears vastly distant. Thus, if one compares the sketch of *any* data point that is close to the original input, to the sketch of the perturbed input the approximation algorithm will return a completely false distance.

To the best of our knowledge this work is the first that *concretely demonstrates* the pitfalls of using common random input r_{cmn} for secure sketching protocols. In this work we focus on distance functions, analogous definitions can be formed for other functions. Note that Definition 2.2 deals with two inputs α and β from distinct users, whereas the following definition deals with the input of a single user and it perturbed version, i.e., α and α^+.

Definition 4.1. *Let $\mathcal{F}_{Approx}$ be the functionality described in Equation (1) for a sketching approach of a distance function d. Let Perturb$(\cdot)$ be the function that adversary $\mathcal{A}$ can apply according to the threat model of Section 3. Let $\alpha \in X$ be a point of the metric space (X, d) with distance function d. Let r_{cmn} be the common random input to the sketching function $\mathcal{S}$. Then we say that Perturb$(\cdot)$ is a* successful (ν, ν')-perturbation attack *for sketching function $\mathcal{S}$ if for any α and r_{cmn}, Perturb(α, r_{cmn}) outputs a point α^+ such that:*

1) *The true distance between α, α^+ is ν, $d(\alpha, \alpha^+) = \nu$,*
2) *The approximate distance between α, α^+ is ν' according to $(\mathcal{S}, \mathcal{G})$ with input r_{cmn}, $\hat{d}_{(\mathcal{S},\mathcal{G})}(\alpha, \alpha^+) = \nu'$,*
3) *The inequality $|\nu' - \nu| > \epsilon\nu$ holds.*

where ϵ is the parameter of the (ϵ, δ) approximation guarantees of $\hat{d}_{(\mathcal{S},\mathcal{G})}$.

One might suggest that it is trivial to mount a successful perturbation attack by generating random data and call it α^+. This naive approach would successfully increase the approximate distance ν' (condition 2), but it would *heavily distort* the original input and as a result the true distance ν would increase as well (i.e., doesn't satisfy their pairwise relation captured in condition 3). Intuitively, for the case where $\nu' > (1 + \epsilon)\nu$, condition 3 guarantees that the perturbed data "appears" more distant from the original than it truly is even when we consider the valid approximation error. For the case where $\nu' < (1 - \epsilon)\nu$, condition 3 guarantees that the perturbed data "appears" more similar from the original than it truly is. In this work we focus on the first case, i.e., $\nu' > (1 + \epsilon)\nu$, thus the adversary wants to hide the high similarity, i.e., small ν, by minimally perturbing the input, i.e. large ν'. Note that due to the triangle inequality, any data point β that is close to α will also appear distant to α^+.

On using Commitment Schemes. It appears that the perturbation attack can be avoided if we force the parties to commit to their data before they receive r_{cmn}. Using a commitment scheme [31] the parties commit to their data, thus any perturbation will be caught due to the binding property of the construction. This mitigation indeed works only if *all data* from *all the users* is available during the initialization of the system and no sketch is created thereafter. In all practical scenarios, however, the system is more dynamic—users join and leave at arbitrary times. One may think that we can accommodate new users by defining epochs where new users can join. This implies that every party must re-compute the sketches from scratch in the beginning of every epoch. This goes against the very reason we used sketching techniques in the first place—to avoid processing the high dimensional data points multiple times.

On using general purpose MPC. Another way of mitigating a perturbation attack would be to use general purpose Multi-Party Computation for both the sketching and the reconstruction steps. This, however, would introduce scalability issues: a party would have to transfer her high-dimensional data *between all the parties* in order to compute

by	$\hat{d}_{Jac} \geq 0.9$									$\hat{d}_{Jac} = 1$								
	$s = 500$			$s = 1,000$			$s = 10,000$			$s = 500$			$s = 1,000$			$s = 10,000$		
κ	d_{Jac}	f_{Success}	Time	d_{Jac}	f_{Success}	Time	d_{Jac}	f_{Success}	Time	d_{Jac}	f_{Success}	Time	d_{Jac}	f_{Success}	Time	d_{Jac}	f_{Success}	Time
10	0.01	1.00	0.01	0.008	1.00	0.03	0.0008	1.00	0.37	0.01	0.98	0.10	0.009	0.99	0.22	0.0009	0.99	2.52
50	0.08	1.00	0.07	0.043	1.00	0.15	0.004	1.00	1.47	0.09	0.95	1.32	0.047	0.96	3.04	0.005	0.98	38.9
100	0.15	1.00	0.14	0.082	1.00	0.30	0.008	1.00	3.00	0.16	0.89	4.07	0.090	0.92	9.23	0.009	0.96	120.1
200	0.27	1.00	0.34	0.159	1.00	0.59	0.018	1.00	5.25	0.28	0.82	12.60	0.166	0.86	27.6	0.019	0.94	380.1

TABLE 1. EVALUATION OF THE PERTURBATION ATTACK ON MINHASH SKETCHES OVER SYNTHETIC DATA. THE TERM κ DENOTES THE SIZE OF THE SKETCH, s IS THE SIZE OF THE SET UNDER ATTACK, f_{SUCCESS} IS THE FREQUENCY OF SUCCESS OF THE PROBABILISTIC ALGORITHM 1. THE DATA POINTS SHOWN ARE THE AVERAGE OVER 5,000 INSTANTIATIONS. TIME IS MEASURED IN SECONDS.

the sketch, therefore making this approach impractical due to its bandwidth requirements. Additionally there is a silent assumption that the rest of the participants are going to be online and have available resources to assist the entire group in sketching, which is unrealistic. Finally, similar to the argument about commitment schemes, when a new user joins the group we have to re-run the MPC protocol for sketching from scratch.

On the level of distortion. Many of the occasions where secure similarity approximation protocols are applied typically employ multiple layers of forensic investigation mechanisms. Therefore in order to *minimize the likelihood of getting detected* (e.g. audit process) the attacker wants to minimize the amount of changes to the input data, i.e. minimize the value of ν in condition 1 of the above definition. Similar to the long line of research on adversarially crafted input for Deep Neural Networks [53], [54], [59], we make the realistic assumption that safety mechanisms are in place to detect radical changes.

Objectives of perturbation attacks. Note that $d_{Jac}(\cdot)$ and $d_{cos}(\cdot)$ as defined in Section 2.3 take values from the range $[0, 1]$. Ideally, a successful (ν, ν')-perturbation attack 1) maximizes the approximate distance $\hat{d}$ so as α and α^+ appear as distant as possible, e.g., $\hat{d}_{Jac}(\alpha, \alpha^+) \approx 1$, while 2) minimizes the true distance between α and α^+, e.g., $d_{Jac}(\alpha, \alpha^+) \approx 0$. We present two such attacks in this section that utilize different tools, namely a randomized algorithm and a non-convex constrained optimization formulation, and provide different guarantees. We slightly abuse notation and indicate by $\hat{d}_{Jac}(\cdot)$ and $\hat{d}_{cos}(\cdot)$ the approximate distance that is returned by a sketching protocol $(\mathcal{S}, \mathcal{G})$.

Incentive to Attack. Users may be inclined to perform such a perturbation attack on several situations. In the case of investigations involving intellectual property theft, industrial espionage, or insider trading, the plaintiff party may want to discover evidence of wrongdoing by matching a set of documents or keywords against a corpus of the defendants data (e.g., emails). In the same spirit, if some form of genetic information is found in a crime-scene the suspect might attempt to masquerade genetic measurements to avoid similarity detection.

4.1. Attacking Minhash Sketches

Minhash sketches are used for approximating the Jaccard distance between sets. We propose a perturbation attack on minhash sketches guaranteed to perform the minimum number of changes to the original input set, thus minimizing $d(\alpha, \alpha^+)$. The perturbation that we apply is in the form of *adding new elements* to the set.

Intuition. The adversary takes as input a set S and the common random input r_{cmn}. The goal is to augment S with the smallest number of new elements in order to create S^+, such that $\hat{d}_{Jac}(S, S^+) = 1$. Recall that the approximate Jaccard distance between two sets is maximized when their κ-dimensional sketches $\sigma()$ differ in all dimensions, i.e., quantity $\hat{d}_{Jac}(\cdot)$ in equation (2) is equal to 1. Thus, the adversary is looking for at most[4] κ new elements such that every dimension of sketch $\sigma(S^+)$ is different from $\sigma(S)$. We denote by t' the number of samples drawn from the metric space. The following algorithm describes the attack, the corresponding proof can be found in the Appendix of this work.

Algorithm 1: Attack Perturb on Minhash Sketches

Input: S, r_{cmn}, κ
Output: S^+ s.t. $\hat{d}_{Jac}(S, S^+) = 1$, $d_{Jac}(S, S^+) = \frac{\kappa}{s+\kappa}$
Use r_{cmn} to sample κ hash functions $(h_1, \ldots, h_\kappa)$;
$\sigma(S) \leftarrow \big(\min_{x \in S}(h_1(x)), \ldots, \min_{x \in S}(h_\kappa(x))\big)$;
$S^+ \leftarrow S$;
for $i = 1$ *to* t' **do**
  Sample an element $z_i \notin S$ uniformly at random;
  for $j = 1$ *to* κ **do**
    if $h_j(z_i) < \min_{x \in S}(h_j(x))$ **then**
      $S^+ \leftarrow S^+ \cup \{z_i\}$;
    end
  end
end

Theorem 4.1. *Let $S \subset [m]$ be the set that is given as an input to Algorithm 1, where $|S| = s$. Let κ be the number of dimensions of the minhash sketch according to equation (2). Then a quasilinear number t' of samples are enough for Algorithm 1 to mount a successful $(1, \frac{\kappa}{s+\kappa})$-perturbation attack for minhash sketching with probability at least*

$$\Pr\big(\{\textit{Succesful Attack}\} | (t' \geq 2c(s+1)\ln^3(s))\big) \geq 1 - \frac{6\kappa c^{1/2}}{s^c}$$

, where c is a positive integer.

Attacking Synthetic Data. We demonstrate the frequency of success and the efficiency of the perturbation

4. There is a case where the same new element of S^+ can contribute to more than one locations of the sketch $\sigma(S^+)$.

attack on synthetic data. We thoroughly tested setups that range across all different variables of the problem: 1) dimension of the sketch $\kappa \in \{10, 50, 100, 200\}$, 2) size of the set under attack $s \in \{500, 1000, 10000\}$, 3) desired mis-approximation $\hat{d}_{Jac}() = 1$ or $\hat{d}_{Jac}() \geq 0.9$. Works such as [43] deploy a sketch of 64 bits to capture similarity of a collection of 8 billion webpages. Therefore, we think that the range of sketches' size 10-200, is indicative of what might be used in practice. The attack is implemented in C++ where the elements of the original set are randomly generated numbers. We used 4-wise independent hash functions. We run 5,000 instantiations for each of the above setups. As observed in Table 1, for the case where the desired approximation is $\hat{d}_{Jac}() \geq 0.9$, the attack succeeds in *all* instantiations, and its execution time is less than one second in most of the parameterizations. In this scenario it is enough for the adversary to discover smaller minhashes for 90% of the κ entries of the minhash sketch. Thus, if there are some small minhash values in the original sketch, the adversary can ignore those and "break" the rest of the sketch, whereas in the case of $\hat{d}_{Jac} = 1$ the adversary is forced to continue searching so as to "break" all κ minhashes. Overall, the frequency of success is extremely high but there are a couple of cases for which the probabilistic guarantees of Theorem 4.1 are not met. One explanation is that the analysis was performed assuming that hash functions are truly random, whereas in the experiment we use 4-wise independent hashing. Table 1 clearly demonstrates that the probabilistic perturbation attack on minhash sketches succeeds in the vast majority of the instantiations and the total time ranges from less than a second to a couple of minutes even when dealing with sets that contain thousands of elements.

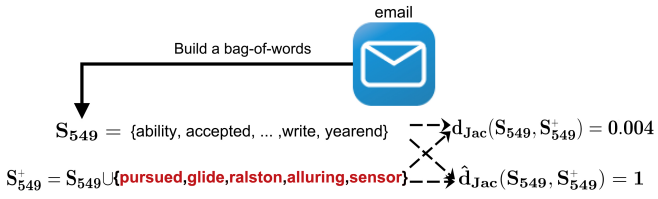

Figure 1. Illustration of the perturbation attack on the email with id 549 from the Enron dataset. The adversary can add the 5 red-colored words in the original email and the approximate distance of our instantiation will be 1 even though the exact distance is 0.004.

Attacking Real Data. To further verify the effectiveness of the attack we tested in real data using the bag of words dataset of Enron emails[5]. We highlight that the findings of the attack on the synthetic data are expected to be similar to those on any real data. This is because the distribution of the hash values used in the attack is not affected significantly by the values of the input set S [6], (as long as there are enough distinct items in the set). In this real dataset every email is transformed into a multiset of words where the stop-words are removed. In this context Jaccard distance captures the similarity between any pair of emails. In our experiment we use the standard Rabin-Karp rolling hash function modulo $n = 105,943$. For simplicity we choose the size of the minhash sketch to be $\kappa = 5$ and the value of c to be 2 (see Theorem 4.1). Without loss of generality, for the purposes of this evaluation we focus on email with id-549 (denoted as set S_{549}), with size $s = 1181$ words, 492 of which are unique. The average time to mount 100 instantiations of the attack was 2.2 seconds. Specifically, 83 out of 100 instantiations mounted successfully a $(1, 0.004)$-perturbation attack and terminated *in less than 1 second*. The remaining 17 instantiations took between 3 to 22 seconds due to the fact that at least one of the minhash values of the original sketch was already too small (< 10). Figure 1 illustrates one of the successful attacks where by adding the 5 words {pursued, glide, ralston, alluring, sensor} in the current email, i.e. create S^{+}_{549}, the approximate distance becomes 1, while the real distance is 0.004. Thus, any future comparison between S^{+}_{549} and a similar email will result in mis-approximation.

Parametrizing the Minhash Attack. In the current description of Algorithm 1 the attacker's goal is to achieve $\hat{d}_{Jac}(S, S^{+}) = 1$, but one can trivially parametrize our algorithm so as to achieve any maliciously chosen output. Specifically, if the goal is to $\hat{d}_{Jac}(S, S^{+}) = \gamma$ then this translates to finding $X/\kappa = \gamma \Rightarrow X = \kappa \cdot \gamma$ elements that substitute the original minhash values in X out of κ dimensions. As an example in our attack on synthetic data we demonstrate the attack for $\gamma = 0.9$.

4.2. Attacking Cosine Sketches

Cosine sketching is used for approximating the cosine distance between vectors. We propose a perturbation attack on cosine sketching guaranteed to output $\hat{d}_{cos}(\cdot) = 1$, while the exact distance between the perturbed and the original vectors depends on the solution of the formulated constrained non-convex optimization problem. The perturbation that we apply is in the form of *adding a new vector* $\vec{x}$ to the original vector $\vec{v}$.

Algorithm 2: Attack Perturb on Cosine Sketch

Input: $\vec{v} \in \mathbb{R}^n, r_{cmn}, \kappa$
Output: $\nu, \vec{v^+} \in \mathbb{R}^n$ s.t. $\hat{d}_{cos}(\vec{v}, \vec{v^+}) = 1$, $d_{cos}(\vec{v}, \vec{v^+}) = \nu$

1 Use r_{cmn} to sample vectors $(\vec{w_1}, \ldots, \vec{w_\kappa})$ from the unit n-sphere
2 Solve the following optimization problem

$$\vec{x} = \underset{\vec{x} \in \mathbb{R}^n}{\text{argmax}} \quad \frac{\vec{v} \cdot (\vec{v} + \vec{x})}{||\vec{v}||_2 ||\vec{v} + \vec{x}||_2}$$

$$\text{subject to} \quad \text{sgn}(\vec{w_i}^T \vec{v}) \cdot (\vec{w_i}^T (\vec{v} + \vec{x})) \leq 0, \; i = 1, \ldots, \kappa.$$

$\nu = d_{cos}(\vec{v}, \vec{v} + \vec{x})$
3 **return** $\nu, \vec{v^+} = \vec{v} + \vec{x}$

Intuition. The adversary takes as input the original vector $\vec{v} \in \mathbb{R}^n$ and r_{cmn}. The goal is to add a new

5. https://archive.ics.uci.edu/ml/datasets/Bag+of+Words
6. S can consist of randomly generated elements (i.e., synthetic data), or words from a real text (e.g., email).

vector $\vec{x}$ to the original $\vec{v}$ in order to create $\vec{v^+}$ such that $\hat{d}_{cos}(\vec{v}, \vec{v^+}) = 1$. Recall that the approximate cosine distance between two vectors is maximized when their κ-dimensional sketches $\sigma()$ differ in all dimensions. Thus the addition of vector $\vec{x}$ to $\vec{v}$ must *change the sign* of the κ inner products with respect to Equation (5) and consequently flip the bits of the sketch $\sigma(\vec{v^+})$. Overall, the adversary wants to maximize the approximate cosine distance, handled by the constraints of the optimization problem, and minimize the exact cosine distance, handled by the objective function of the optimization.

In Algorithm 2 the function $\text{sgn}(x)$ has output -1 in case $x < 0$ and output $+1$ in case $x \geq 0$. The unit n-sphere is defined as the set of points $\{u \in \mathbb{R}^{n+1} : ||u|| = 1\}$. Notice that minimizing the exact cosine distance is equivalent to maximizing the cosine similarity as it is described in Equation (4), so our problem is formed as a maximization of the cosine similarity $C(\vec{v}, \vec{v} + \vec{x})$. Algorithm 2 requires to solve a non-convex, non-linear, high-dimensional constrained optimization problem. Furthermore the objective function presents discontinuity at point $\vec{x} = -\vec{v}$, see Figure 2. Since closed form solutions are generally challenging for this setup, we approximate the solution of the above problem using iterative algorithms from standard optimization toolboxes. Figure 2 visualizes the objective function for the case where $v \in \mathbb{R}^2$.

Theorem 4.2. *Let $\vec{v} \in \mathbb{R}^n$ be the vector that is given as an input to Algorithm 2. Let also $\vec{w_i} \in \mathbb{R}^n$ be a vector sampled from the unit n-sphere using r_{cmn} according to Algorithm 2, where $i = [1, \kappa]$. Then, Algorithm 2 is a successful $(1, \nu)$-perturbation attack for cosine sketching.*

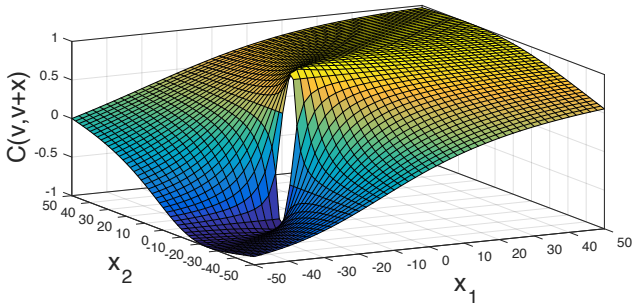

Figure 2. An illustration of the objective function of the maximization problem of Algorithm 2 where $n = 2$ and $\vec{v} = (20, 10)$. The X-,Y-axis denote the x_1 and x_2 dimension of vector to be added, $\vec{x}$.

Attacking Synthetic Data. We evaluated the performance of the attack on synthetic data using the interior point algorithm of MATLAB [1] where the input vector $\vec{v}$ is a randomly generated vector from $\mathbb{R}^n$. We thoroughly tested setups that range across the different variables of the problem: 1) the number of dimensions of the vector under attack that takes values $n \in \{500, 1000, 5000\}$, and 2) the size of the sketch under attack that takes values $\kappa \in \{10, 50, 100, 200\}$. To generate vectors $\vec{w_i} \in \mathbb{R}^n$ we sampled vectors from the n-sphere of unit radius centered at the origin. We run the above setups with 10 different common randomness inputs r_{cmn} and present the mean. As one may observe in Table 2, the approximate distance is always $\hat{d}_{cos}() = 1$ which implies that all the returned solutions were part of the feasible region of the optimization problem. The value of the exact cosine distance d_{cos} between the original and the perturbed data depends on the returned solution of the optimization problem. Note that different solution methods can potentially result in even lower d_{cos} values. Depending on the optimization toolbox and the number of dimensions the time performance may vary, in our case all the executions terminated within a couple of minutes.

	$n = 500$		$n = 1,000$		$n = 5,000$	
κ	d_{cos}	$\hat{d}_{cos}$	d_{cos}	$\hat{d}_{cos}$	d_{cos}	$\hat{d}_{cos}$
10	0.005	1	0.002	1	0.0005	1
50	0.02	1	0.01	1	0.002	1
100	0.05	1	0.02	1	0.005	1
200	0.11	1	0.06	1	0.01	1

TABLE 2. EVALUATION OF THE PERTURBATION ATTACK ON COSINE SKETCHES OVER SYNTHETIC DATA. THE DATA POINTS SHOW THE AVERAGE VALUE OVER 10 INSTANTIATIONS.

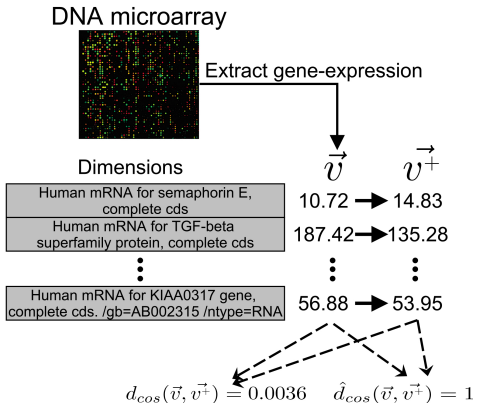

Figure 3. Illustration of the perturbation attack on a gene-expression of an adenoma patient. If the adversary perturbs the vector $\vec{v}$ of 7086 dimensions to create $\vec{v^+}$ then the approximate cosine distance will be 1 even though the exact distance is 0.0036.

Attacking Real Data. We demonstrate the perturbation attack of Algorithm 2 on a real dataset[7] of human gene-expression levels that can be found in the work of Notteramn *et al.* [51]. The authors perform a clustering analysis on the vectors of gene-expression levels so as to capture similarity patterns between healthy patients, patients with adenoma and patients with adenocarcinoma. It is rather common to perform similarity-based analysis on genomic data with the goal of understanding and diagnosing diseases at the molecular level. We highlight that the findings of the attack on the synthetic are expected to be similar to those on any real data. This is because the generative model of the input vector $\vec{v}$ does not affect the sign of the inner product with a random vector $\vec{w}$.

We approximate the solution of the optimization problem using the interior point algorithm from MATLAB [1].

7. http://genomics-pubs.princeton.edu/oncology/

We use a cosine sketch of $\kappa = 100$ dimensions and we repeat the experiment for 10 different initializations of the vectors $(\vec{w_1}, \ldots, \vec{w_\kappa})$. The input vector is denoted as $\vec{v}$ and it has $n = 7,086$ dimensions each of which is a gene-expression measured with a DNA microarray. We report that all of the instantiations successfully satisfied the optimization constraints and thus resulted in $\hat{d}_{cos}(\vec{v}, \vec{v^+}) = 1$. The average ν value was 0.0033 with a maximum value of 0.0039. Therefore, on average we mounted a successful $(1, 0.0033)$-perturbation attack. One of the recorded instantiations is illustrated in Figure 3 where it shows that if the adversary perturbs $\vec{v}$ to form $\vec{v^+}$ then according to the cosine sketching initialization we have $\hat{d}_{cos}(\vec{v}, \vec{v^+}) = 1$, even though their exact cosine distance is $d_{cos}(\vec{v}, \vec{v^+}) = 0.0036$.

Parametrizing the Cosine Attack. In Algorithm 2 the attacker's goal is to achieve $\hat{d}_{cos}(\vec{v}, \vec{v^+}) = 1$, but one can trivially parametrize our algorithm so as to achieve any maliciously chosen output. In case the goal of the attacker is to achieve $\hat{d}_{cos}(\vec{v}, \vec{v^+}) = \gamma$ then this is equivalent to reducing the number of constraints of the optimization problem from κ to $X/\kappa = \gamma \Rightarrow X = \kappa \cdot \gamma$. The attacker can choose either deterministically or randomly which X out of the κ constraints to keep.

Future Directions. There are several interesting future directions in regard to the area of perturbation attacks. From the adversary's perspective, it is interesting to attack more sketching techniques as well as investigate *context-aware* perturbation attacks. Under this more focused type of attacks the adversary perturbs the data in a way that is relevant to the semantics of the data. For instance, if the first dimension of the vector under attack represents "age" then it is better for the adversary not to change it to a negative value. Similarly if the email under attack comes from a bank then adding sentences from a cooking recipe as part of the perturbation might reveal that something is wrong. This family of attacks can be captured by *additional constraints* so as to form perturbed data that look more realistic in the particular context.

In our work we focused on perturbation mechanisms that *add* information. In a different setup the adversary might choose to remove existing information or transform small pieces of data to something equivalent, e.g., in the case of emails, phrases that have the same meaning.

From the defender's perspective a new framework should be devised in order to mitigate perturbation attacks. The attacks proposed in this paper exploit the fact that the common random input r_{cmn} of Definition 2.2 is known to all the parties before the protocol begins. The goal of the next Section is to propose a new architecture that allows secure sketching *without revealing* the common random input to the participating users.

5. Server-Aided Approximation

In this Section we reframe the architecture of secure sketching protocols so that we can 1) still use the well-studied sketching techniques based on the common random input r_{cmn}, and 2) *eliminate* the possibility of an offline perturbation attack. In our proposed *server-aided* design we introduce a new semi-honest entity, i.e., the server S, that has exclusive access to the common random input r_{cmn} and assists in the sketching protocols. Compared to previous approaches, the main difference of our design is that a client *does not have direct access* to the common random input. The sketching function that was previously a local computation (as described in Section 3), is replaced by a two-party protocol denoted as Sketching between the server and the client. We capture the new *functionality* as follows:

Functionality $\mathcal{F}_{\text{S-approx}}$

- *Input:* Party C_A provides v_A, party C_B provides v_B, party S provides r_{cmn}.
- *Output:* All three parties receive $\hat{d}(v_A, v_B)$.

Notice that in case client C_A (similarly for client C_B) observes the values of σ_A, then it is possible for the C_A to infer r_{cmn}, which is an attack that defeats the purpose of the server-aided model. For example, consider the case where r_{cmn} is used to sample k-independent hash functions and the values of σ_A consists of the evaluations of the above hash functions. An adversary that observes the hash values can easily infer the coefficients of the hash function by solving a system of equations [35]. In our design, protocol Sketching is executed between a client and the server and it returns the *encrypted* sketch σ_A to C_A so as to avoid the above type of attacks.

The real model. Let Π be a three-party protocol computing the functionality $\mathcal{F}_{\text{S-approx}}$. For ease of exposition we consider the execution of Π in the presence of an adversary $\mathcal{A}$ as being coordinated by a nonuniform environment $\mathcal{Z} = \{\mathcal{Z}_\lambda\}$, much like [14], [37]. In the beginning $\mathcal{Z}$ gives input $(1^\lambda, v_A)$ to C_A, input $(1^\lambda, v_B)$ to C_B, and gives z and X to $\mathcal{A}$, where z denotes an auxiliary input and $X \in \{C_A, C_B, S\}$ is the corrupted party. At this point the parties interact with each honest party behaving as instructed by Π. At the end of the protocol, adversary $\mathcal{A}$ gives to $\mathcal{Z}$ an output which is an arbitrary function of $\mathcal{A}$'s view. Additionally, $\mathcal{Z}$ gets the output of the honest parties. Finally, environment $\mathcal{Z}$ outputs a bit. We denote as $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda)$ the random variable that represents the value of this bit.

The ideal model. In this model there is a trusted party that computes $\mathcal{F}_{\text{S-approx}}$ on behalf of the parties. Similar to the real model, environment $\mathcal{Z}$ gives inputs $(1^\lambda, v_A)$ and $(1^\lambda, v_B)$ to parties C_A and C_B, respectively; it also gives gives z and X to $\mathcal{A}'$ where $X \in \{C_A, C_B, S\}$ indicates the corrupted party. All the parties send their input to the trusted party. The trusted party computes $\mathcal{F}_{\text{S-approx}}$ and sends $\hat{d}(v_A, v_B)$ to all the parties. In the next step $\mathcal{A}'$ outputs to $\mathcal{Z}$ an arbitrary function of the view of $\mathcal{A}'$. The honest parties also give their output to $\mathcal{Z}$. As a final step $\mathcal{Z}$ outputs a bit. We denote as $\mathsf{IDEAL}_{\Pi,\mathcal{A}',\mathcal{Z}}(\lambda)$ the random variable that represents the value of this bit.

Definition 5.1. *Let* Π *be a three-party protocol for comput-*

Protocol	C-Input	S-Input	C-Output	S-Output	Summary
PrvComparison*	a	b	-	$[t]$	[t=1] if $a < b$, [0] otherwise
EncComparison*	$SK_P^{(C)}, SK_{GM}^{(C)}, l$	$[a], [b], l$	t	-	t=1 if $a < b$, 0 otherwise
EncComparison2*	$SK_P^{(C)}, SK_{GM}^{(C)}, l$	$[a], [b], l$	-	$\|t\|$	$\|t = 1\|$ if $a < b$, $\|0\|$ otherwise
ChangePartyEnc*	$SK_{GM}^{(C)}$	$SK_{GM}^{(S)}, \|b\|$	$\|b\|$	-	Encrypts $\|b\|$ under $SK_{GM}^{(S)}$
kIndHashing	$SK_P^{(C)}, x, k, p$	$\{a_i\}_{i=0}^{k-1}, p$	-	$[h]$	$[(\sum_{i=0}^{k-1} a_i x^i) \mod p]$
EncHashing*	$SK_P^{(C)}, k, p$	$[x], \{a_i\}_{i=0}^{k-1}, p$	-	$[h]$	$[(\sum_{i=0}^{k-1} a_i x^i) \mod p]$
FindMin	$SK_{GM}^{(C)}, SK_P^{(C)}, l$	$\{[y_i]\}_{i=1}^{n}, l$	-	$[min]$	$[\min_i y_i]$
UpdateOddSketch	$SK_{GM}^{(C)}, SK_P^{(C)}, SK_{DGK}^{(C)}, u, k$	$[x], \{a_i\}_{i=0}^{k-1}, u, (\|skt_0\|, \ldots, \|skt_{u-1}\|)$	-	$(\|skt'_0\|, \ldots, \|skt'_{u-1}\|)$	Update odd sketch with x
Sketching-Cosine	$\vec{v}, SK_P^{(C)}, SK_{GM}^{(C)}$	$\{\vec{w}_i\}_{i=1}^{\kappa}, SK_{GM}^{(S)}$	$(\|\sigma_1\|, \ldots, \|\sigma_\kappa\|)$	-	Encr. cosine signature
Sketching-Odd	$S, k, u, SK_{GM}^{(C)}, SK_P^{(C)}, SK_{DGK}^{(C)}$	$\{h_i^{min}\}_{i=1}^{\kappa}, h_{odd}, p, SK_P^{(S)}, SK_{GM}^{(S)}$	$(\|\sigma_1\|, \ldots, \|\sigma_\kappa\|)$	-	Encr. odd-minhash signature

TABLE 3. AN OVERVIEW OF THE PROTOCOLS. FOR BREVITY WE ASSUME THAT THE PUBLIC KEYS OF THE SERVER $PK_P^{(S)}, PK_{GM}^{(S)}$ AND THE CLIENT $PK_P^{(C)}, PK_{GM}^{(C)}, PK_{DGK}^{(C)}$ ARE PUBLICLY AVAILABLE AND THUS NOT PASSED AS AN INPUT TO THE PROTOCOLS.

ing $\mathcal{F}_{\text{S-approx}}$ functionality. We say that Π securely computes *$\mathcal{F}_{\text{S-approx}}$ in the presence of semi-honest adversaries corrupting one party if for any PPT semi-honest adversary $\mathcal{A}$ there exists a PPT semi-honest adversary $\mathcal{A}'$ such that, for every polynomial size circuit family $\mathcal{Z} = \{\mathcal{Z}_\lambda\}$ corrupting at most one party, the following is negligible:*

$$|\Pr[\mathit{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda) = 1] - \Pr[\mathit{IDEAL}_{\Pi,\mathcal{A}',\mathcal{Z}}(\lambda) = 1]|.$$

Notice that if the adversary were to corrupt both a client and the server then she would have access to the common random input, and thus become capable of mounting a perturbation attack. We note here that the server-aided approach has been successfully deployed [38], [39] in various other problems. The proposed perturbation attacks of the previous Section are based on the fact that all clients have offline and direct access to the common random input r_{cmn}. Under our server-aided design an adversary can only attempt an *online* attack, hoping to infer the r_{cmn} from the value of $\hat{d}(\cdot)$, by performing a series of Sketching and Reconstruct executions. Using rate-limiting techniques (e.g., [39]) one can mitigate such an online attack. This scenario, however, is beyond the scope of this paper.

Composition of Building Blocks. We define separate building blocks that can be combined and the proof of security for the overall construction can be derived using modular composition [13]. The model is called *hybrid model with ideal access to functions* $f_1, \ldots, f_m$ or simply $(f_1, \ldots, f_m)$-hybrid model . In the real life experiment we assume the existence of an incorruptible trusted party T for evaluating $f_1, \ldots, f_m$; all parties hand their input to T and they receive the corresponding output. As a next step, the ideal evaluation of f at each step is replaced with the invocation of a protocol—we refer the reader to [13] for a detailed exposition. In case the function returns an encrypted output, a party passes a public key as an input and we assume that the necessary encryption algorithm is hardwired to the corresponding function. Table 3 summarizes all the two-party protocols, which in our case are executed between the server and the client. Using the above building blocks we construct a secure two-party analogue for minhashing (via odd sketches) and cosine skething. Due to lack of space, protocols that are marked with $*$ (simple modification of already proposed protocols [4], [10], [61] or new protocols) can be found in the Appendix of this work.

5.1. Building Blocks

k-Independent Hashing over Encrypted Data. The functionality of $\mathcal{F}_{\text{kIndHash}}$ is as follows. The input of the server is the set of parameters of a k-independent hash function—i.e., the coefficients $\{a_i\}_{i=0}^{k-1}$ and the prime p of a $(k-1)$ degree polynomial on $\mathbb{Z}_p$. The client has the input x which is used to evaluate the polynomial on $\mathbb{Z}_p$. The degree of the polynomial as well as the modulo p are considered to be known to both parties. At the end of the protocol the server receives the evaluation of the polynomial $a_0 + a_1 x + \ldots + a_{k-1} x^{k-1} \mod p$ that is encrypted with the client's public key. We do not use a private polynomial evaluation technique due to the fact that we require the output to be *encrypted*. The server should not learn any information about the client's input x and the client should not learn any information about the coefficients $\{a_i\}_{i=0}^{k-1}$ of the polynomial. A more thorough exposition of the protocol is provided in the Appendix of this work.

Lemma 5.1. *Protocol kIndHash correctly and securely computes $\mathcal{F}_{\text{kIndHash}}$ in the ($\mathcal{F}_{\text{PrvComp}}$)-hybrid model.*

Update Encrypted Odd Sketch. The functionality of $\mathcal{F}_{\text{UpdateOddSketch}}$ is as follows. The input of the server consists of i) the bits of an odd sketch $(skt_0, \ldots, skt_{u-1})$ encrypted with the client's public key, ii) the parameters of the $(k-1)$-degree polynomial that is used as the hash function h_{odd}, and iii) the input x of the polynomial encrypted with client's public key. The input of the client is the set of secret keys. At the end of the protocol the server receives an updated odd sketch where the bit in location $h_{odd}(x)$ of the sketch is flipped, while the client receives no output. The server and the client should not learn which bit of the odd sketch is flipped or the input x of the polynomial. One novel idea of our design is the use of DGK with message space $\mathbb{Z}_u$, where u is also the length of the sketch, so as to securely translate the hash value into a bit-mask, and eventually apply the mask to the original sketch. A thorough overview of the protocol is provided in the Appendix of this work.

Lemma 5.2. *Protocol UpdateOddSketch correctly and securely computes $\mathcal{F}_{\text{UpdateOddSketch}}$ in the ($\mathcal{F}_{\text{EncHashing}}$, $\mathcal{F}_{\text{ChangeEnc}}$)-hybrid model.*

Find Minimum over Encrypted Values The function-

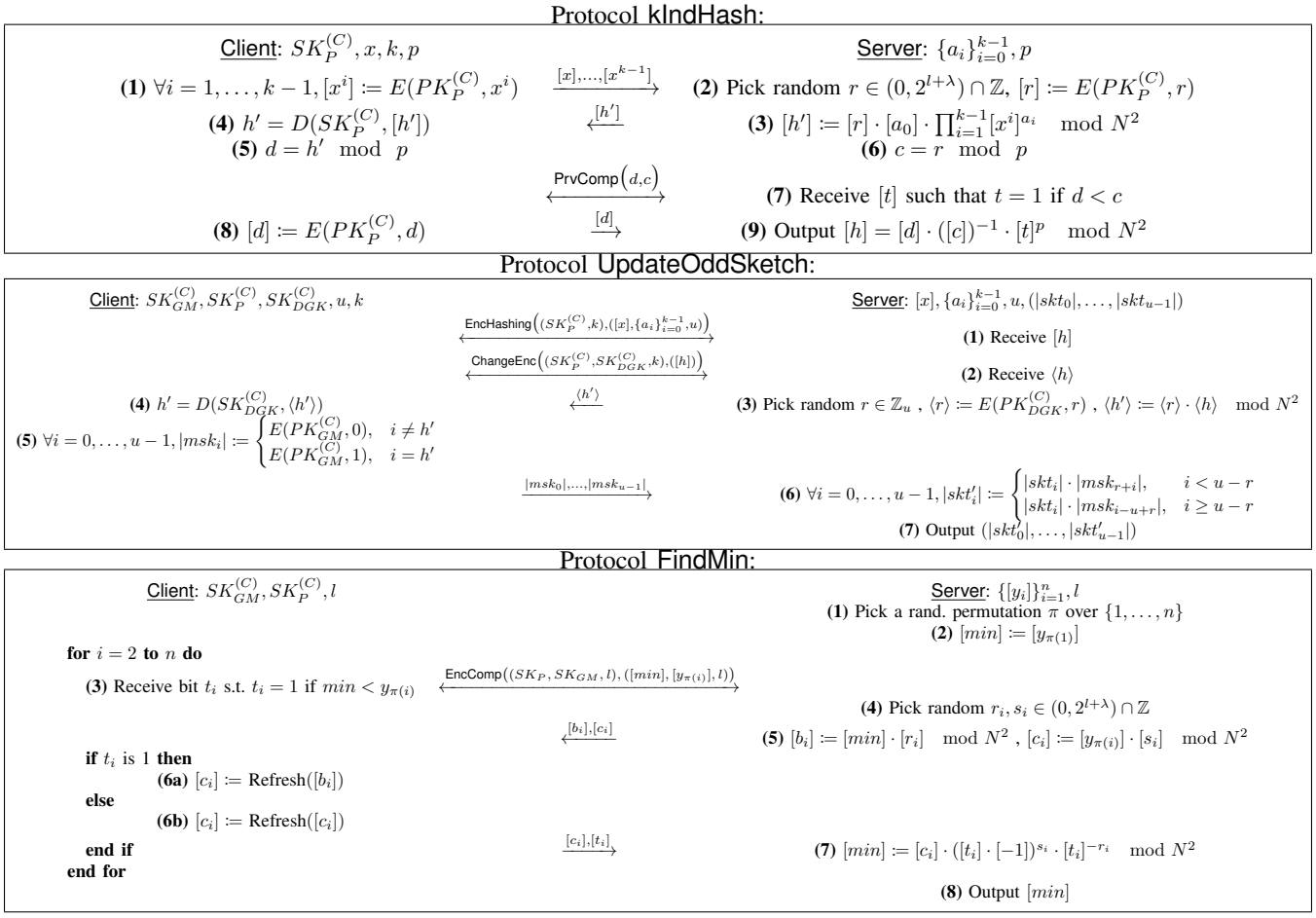

Figure 4. Two-party protocols between a client and the server that are used as building blocks for Sketching.

ality of $\mathcal{F}_{\text{FindMin}}$ is as follows. The input of the server is values $y_1, \ldots, y_n$ that are encrypted with the public key of the client. The client's input is its secret key. At the end of the protocol the server's output is the minimum value of the set $\{y_1, \ldots, y_n\}$ which is encrypted with the client's public key. The server should learn neither which of the original ciphertexts correspond to the minimum value, nor any information about the underlying values of the plaintexts. The client should not learn the values of the underlying plaintexts. Our protocol is following the footsteps of the argmax protocol presented in [10].

Lemma 5.3. *Protocol FindMin correctly and securely computes $\mathcal{F}_{FindMin}$ in the ($\mathcal{F}_{EncComp}$)-hybrid model.*

5.2. Protocols for the Server-Aided Model

Approximating Jaccard Distance via Odd Sketches We employ the protocols of the previous subsection as building blocks to *securely* approximate Jaccard distance using the construction by Mitzenmacher *et al.* [47]. As denoted in Figure 6, the input of the server consists of the set of κ minhash functions $\{h_i^{min}\}_{i=1}^{\kappa}$, the hash function for the creation of the odd sketch h_{odd}, as well as the corresponding secret keys. Recall that $\{h_i^{min}\}_{i=1}^{\kappa}$ and h_{odd} are generated using the common randomness r_{cmn} that can only be accessed by the server. The input of the client consists of her data (represented by the set of elements $\{e_j\}_{j=1}^{n}$), as well as the publicly known moduli p, u, and the secret keys. At the end of the protocol the client receives the odd sketch encrypted with the server's public key.

Lemma 5.4. *Protocol SketchingOdd correctly and securely computes $\mathcal{F}_{SketchingOdd}$ in the ($\mathcal{F}_{kIndHashing}$, $\mathcal{F}_{FindMin}$, $\mathcal{F}_{UpdateOddSketch}$, $\mathcal{F}_{ChangePartyEnc}$)-hybrid model.*

Approximating Cosine Distance via Cosine Sketching We approximate cosine distance as follows. The input of the server consists of the vectors $\vec{w}_i$, that are sampled uniformly at random from the n-dimensional unit hypersphere. The input of the client consists of her data which is represented by the vector $\vec{v}$. Note that vectors $\vec{w}_i$ are generated using the common randomness r_{cmn} that can only be accessed by the server. At the end of the protocol the client receives the cosine sketch encrypted with the server's public key.

Lemma 5.5. *Protocol SketchingCosine correctly and securely computes $\mathcal{F}_{SketchingCosine}$ in the ($\mathcal{F}_{EncComparison2}$,*

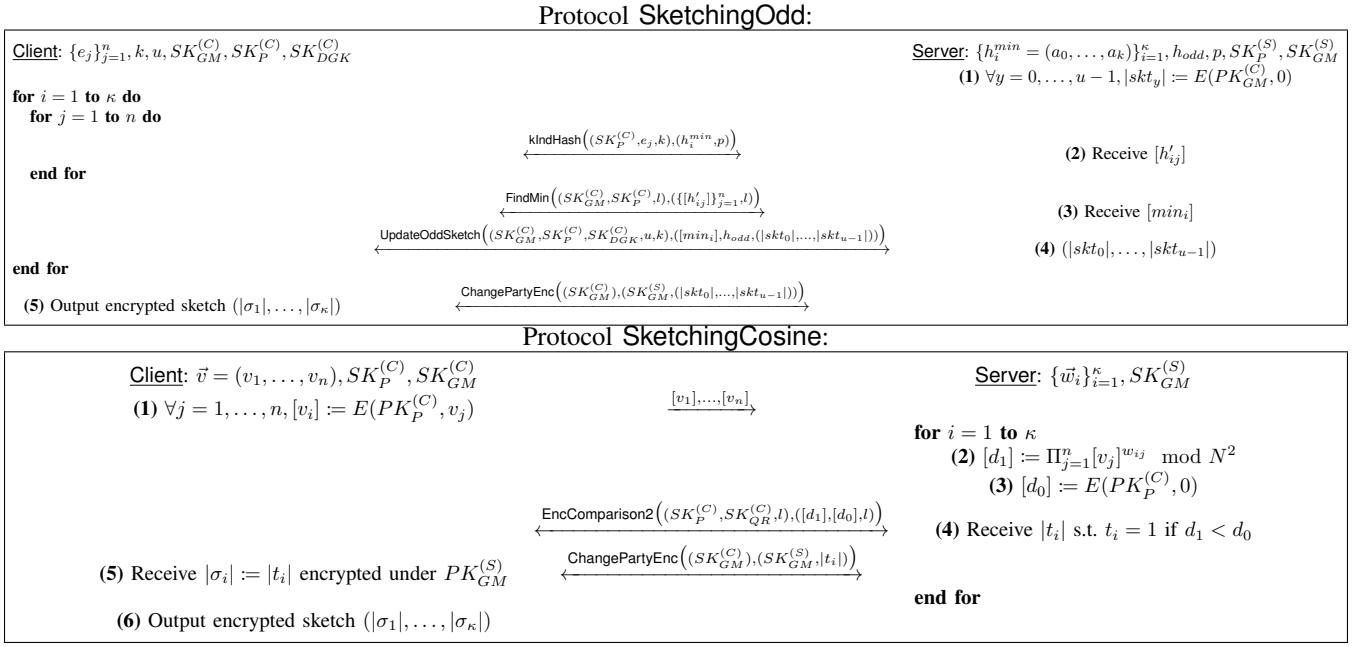

Figure 5. The sketching protocols between the server and the client for the server-aided model.

$\mathcal{F}_{ChangePartyEnc}$)-hybrid model.

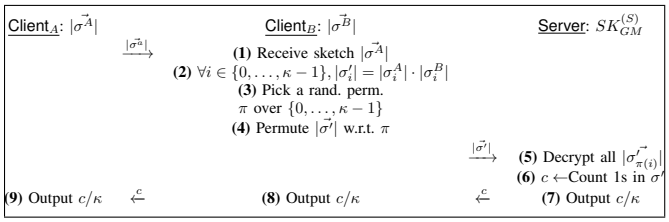

Figure 6. The reconstruction of SketchingCosine between the server and the clients. The reconstruction for SketchingOdd is the same for steps **(1)**-**(6)**; steps **(7)**, **(8)** follow the reconstruction of Equation (3).

Reconstruct Protocol The power of the sketching techniques that we chose for approximating Jaccard distance and cosine distance lies in the fact that their reconstruction function is simple and efficient. Both techniques follow the same reconstruction process which essentially performs an exclusive-or operation between the two sketches, and then counts the number of 1 values (see Equations (3) and (5)). Taking advantage of the homomorphic properties of the Goldwasser-Micali cryptosystem we build very efficient Reconstruct protocols.

Lemma 5.6. *Protocol Reconstruct is correct and secure in the semi-honest model.*

On the Choice of Building Blocks. Since our protocols follow a modular design, one can substitute the proposed building blocks with protocols that follow other MPC techniques so as to further optimize the performance of our constructions. The work presented in this paper is meant to present to principles of this modular design and is not representative of a highly-optimized implementation. According to the work of Bost *et al.* [10], comparison protocols that utilize specialized homomorphic cryptosystems [26], [61] are more efficient when the input is encrypted. Thus, our implementation invokes variations of the above protocols, namely EncComp and EncComp2. For the comparison protocol on unencrypted inputs, Bost *et al.* [10] denote that a garbled circuit approach [6] results in a more efficient implementation. In our implementation we followed the work of Veugen [61], and therefore one can further speedup our implementation by invoking a garbled circuit design instead. We note that well-known protocols that are purely based on garbled circuits for functionality such as FindMin can not be deployed because the input of the FindMin is a set of *encrypted* inputs (see Table 3). A similar argument holds for the output of kIndHashing which is encrypted. Thus, to the best of our knowledge, the most promising speedup opportunity would be opting for garbled circuit designs for the simplest building blocks, such as comparison.

6. Evaluation

Implementation Setup We implemented the proposed protocols in C++ using existing libraries as well as newly implemented building blocks. For serializing the communication between the server and client we use Protocol Buffers [34]. All the arithmetic operations are performed with the gmp multiple precision library [24]. We use the Advanced Crypto Software Collection [7] implementation of the Paillier cryptosystem, and an open-source implementation of the GM cryptosystem. We implemented the DGK cryptosystem in C++ following the design principles of [7] and the directions of the original work [20], [21].

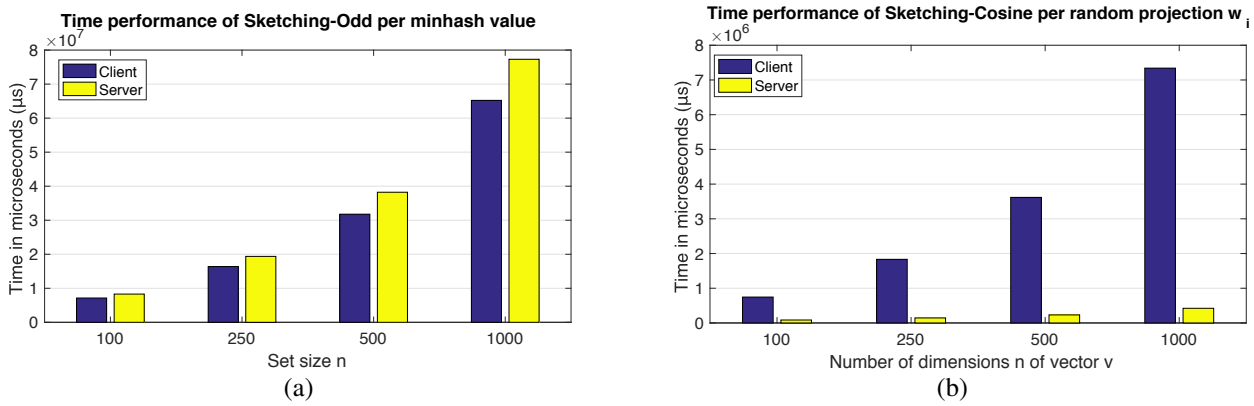

Figure 7. Subfigure (a): Time performance for varied set size of the SketchingOdd protocol. Time averaged for a single minhash over five runs. Subfigure (b): Time performance for varied number of vector dimensions of the SketchingCosine protocol. Time averaged for a single random projection over five runs.

For the minhashing via odd sketching protocols we choose the security parameter $\lambda = 100$. Given the scale of our experiments the k-independent hashing setup is the following: we choose $k = 4$ and a prime p that is at least an order of magnitude larger than the size of the set—i.e., $p \geq 10n$. As explained in the description of protocol UpdateOddSketch, prime u of the DGK cryptosystem is set to have the same value as the length of the odd sketch. As it is also noted in [10] the parameterization of Paillier has to be such that the homomorphic operations do not overflow the message space. To accomplish this instantiation we analyze the two phases of the protocol. The first phase is the kIndHashing computation; let l' be the maximum bit-length of the inputs x. In step **(1)** of protocol kIndHashing involves $(k-1)$ exponentiations among which the plaintext x^{k-1} can have the maximum length of $l'_{\max} = (k-1)l'$ bits. Step **(3)** of protocol kIndHashing involves $(k-1)$ multiplications and $(k+1)$ additions of numbers that are at most $l'_{\max}$ bits long. Therefore it is sufficient for N to be such that $\log N \geq (k^2-k-2)(l'/2)+2+\lambda$. After the execution of kIndHashing the numbers involved in protocols PrvComparison and EncComparison are $\log p$ bits long, since they are hash values. Thus protocols PrvComparison and EncComparison operate on integers that are at most $l = \log p$ bits long. Consequently, it is sufficient for N to be such that $\log N > \log p + \lambda + 1$. We satisfy the above inequalities by choosing $\log N \geq 1024$.

Regarding the protocols for cosine sketching, we also choose a security parameter $\lambda = 100$. Recall that vectors $\vec{w_i} = (w_{i1}, \ldots, w_{in})$ are sampled uniformly at random from the n-dimensional hypersphere, so each value w_{ij} is a real number. We can transform the above real numbers to integers by multiplying with a constant K and rounding, allowing us to interpret w_{ij} as part of Paillier's message space. The purpose of the random projection is to compute the sign of the inner product thus one can choose a relatively small K. In our implementation we choose $K = 1000$. Similarly to the previous instantiation, the parameterization of Paillier should not overflow by the homomorphic operations of the encrypted inner product that is performed in step **(2)** of protocol Sketching-Cosine. Let l be the maximum length in bits of the entries in $\vec{v}$. Then step **(2)** of protocol Sketching-Cosine involves the multiplication of a $\log K$ bit long integer with an l bit long integer. Thus, it is sufficient for N to be such that $\log N \geq \log K + l + n$. Finally, in our implementation, both GM and DGK have moduli that are at least 1024 bits long. The implementation of the protocols and the serialization of the server is around 1400 lines, while the code for the client is around 1100 lines, measured using cloc tool.

Performance. We evaluate the scalability of the server-aided design based on the described implementation setup. In Figure 7 we present the recorded computation time for the sketching protocols.

Protocol	n			
	100	250	500	1000
Sketch-Odd	1112 KB	2930 KB	6218 KB	13201 KB
Sketch-ShimHash	69 KB	165 KB	324 KB	644 KB

TABLE 4. COMMUNICATION OVERHEAD OF SKETCHING. AVERAGE OVER FIVE RUNS.

The client and server have similar time performance for the SketchingOdd protocol. This is mostly due to the fact that both parties are subject to a slowdown by a similar number of encrypt/decrypt operations. The time performance presented in Figure 7-(a) is for a a single minhash value (i.e., $\kappa = 1$), and an odd sketch of 151 bits (i.e., $u = 151$). Note that the computational overhead scales linearly with κ: for $\kappa > 1$ we have the same computational overhead as the one depicted in Figure 7-(a), only κ times larger. Notice, however, that the computation for each of the κ dimensions of the sketch is *independent* of each other, thus the overall task is parallelizable. On the other, hand the computational overhead of the client in protocol SketchingCosine is significantly higher than the one of the server. This is mainly caused by the encryption of each dimension of $\vec{v}$, which translates to a large number of exponentiations taking place in step **(1)** of the protocol. Furthermore, the performance of the server (time) is measured when we have a single random projection, i.e., $\kappa = 1$, thus steps **(2)**-**(4)**

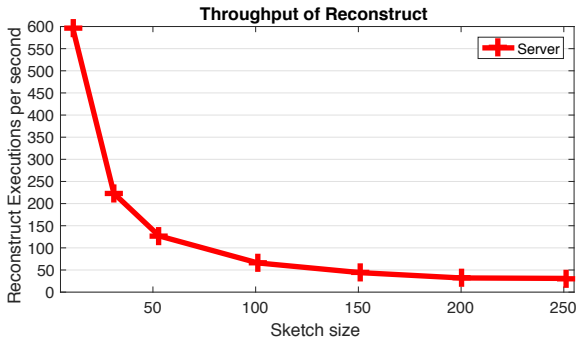

Figure 8. Throughput of the server Reconstruct protocol for varied sketch sizes. Average values over five runs.

of SketchingCosine are repeated only once. Similar to the case of SketchingOdd, for $\kappa > 1$ the overall task is highly parallelizable into κ tasks. The communication overhead of the sketching protocols for various values of n is depicted in Table 4.

In our design we prioritize the *speedup the reconstruction protocol*, since it is the protocol that is executed multiple times throughout the lifetime of the system—once for every pairwise approximation. On the contrary, the sketching protocol is invoked only once for every high-dimensional data point, so as to create the sketch. Thus, using odd sketches (rather than regular minhashing) introduced, indeed, some overhead in the overall sketching protocol but resulted in a fast and more scalable reconstruction protocol. Generally, the reconstruction protocol from the server's perspective is the same, regardless of whether we are approximating Jaccard or cosine similarity, since the only task performed by the server is to decrypt κ ciphertexts encrypted under GM. The end result is a rather scalable performance illustrated in Figure 8.

7. Conclusion

In this paper we introduced the notion of perturbation attacks for secure similarity approximation protocols. We proposed concrete perturbation attacks for the well-studied minhash and cosine sketching techniques, and measured the performance and scalability of the attacks on both real and synthetic data, using various parameters.

Subsequently, we formally defined a server-aided model that mitigates the aforementioned attacks. We also proposed new sketching protocols for this new architecture, building upon state-of-the-art sketching techniques. Our design and implementation aimed at speeding up the reconstruction protocols, as they constitute the part of the overall computation that is executed most frequently—thus having the most severe impact on overall performance. We evaluated the implementation of the proposed protocols and demonstrated that this architecture achieves the desired scalability for the reconstruction process, with reasonable performance.

Acknowledgments

The authors would like to thank Seny Kamara for his insightful comments on the proofs of Section 5. This work was done while the first author was visiting Symantec Research Labs in Culver City, CA. Since then, the first author was partially supported by the U.S. National Science Foundation under grants CNS–1228485, and CNS–1525044, and by the Kanellakis Fellowship at Brown University.

References

[1] "MATLAB and Optimization Toolbox Release 2016a, The Mathworks, Inc., Natick, Massachusetts, United States." 2016.

[2] M. Abadi, D. Boneh, I. Mironov, A. Raghunathan, and G. Segev, "Message-locked encryption for lock-dependent messages," in *CRYPTO*, 2013, pp. 374–391.

[3] P. Baldi, R. Baronio, E. De Cristofaro, P. Gasti, and G. Tsudik, "Countering gattaca: Efficient and secure testing of fully-sequenced human genomes," in *ACM CCS* , 2011, pp. 691–702.

[4] F. Baldimtsi and O. Ohrimenko, "Sorting and searching behind the curtain," in *FC*, 2015, pp. 127–146.

[5] M. Barreno, B. Nelson, A. D. Joseph, and J. D. Tygar, "The security of machine learning," *Machine Learning*, vol. 81, no. 2, pp. 121–148, 2010.

[6] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, "Efficient garbling from a fixed-key blockcipher," in *IEEE S&P*, 2013, pp. 478–492.

[7] J. Bethencourt. (2010) Advanced Crypto Software Collection. [Online]. Available: http://acsc.cs.utexas.edu/libpaillier/

[8] M. Blanton and P. Gasti, "Secure and efficient protocols for iris and fingerprint identification," in *ESORICS*, 2011, pp. 190–209.

[9] C. Blundo, E. De Cristofaro, and P. Gasti, "EsPRESSO: Efficient privacy-preserving evaluation of sample set similarity," *J. Comput. Secur.*, vol. 22, no. 3, pp. 355–381, 2014.

[10] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine Learning Classification over Encrypted Data," in *NDSS*, 2015.

[11] A. Z. Broder, "On the resemblance and containment of documents," in *In Compression and Complexity of Sequences (SEQUENCES)*, 1997, pp. 21–29.

[12] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," *J. Comput. Syst. Sci.*, vol. 60, no. 3, pp. 630–659, 2000.

[13] R. Canetti, "Security and composition of multiparty cryptographic protocols," *J. Cryptology*, vol. 13, no. 1, pp. 143–202, 2000.

[14] ——, "Security and composition of cryptographic protocols: A tutorial (part i)," *SIGACT News*, vol. 37, no. 3, pp. 67–92, Sep. 2006.

[15] N. Carlini, P. Mishra, T. Vaidya, Y. Zhang, M. Sherr, C. Shields, D. Wagner, and W. Zhou, "Hidden voice commands," in *Proc. of 25th USENIX Sec. 16*, Aug. 2016, pp. 513–530.

[16] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *STOC*, 2002, pp. 380–388.

[17] Y. Chen, B. Peng, X. Wang, and H. Tang, "Large-scale privacy-preserving mapping of human genomic sequences on hybrid clouds," in *NDSS*, 2012.

[18] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, Apr. 2005.

[19] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma, "Adversarial classification," in *Proc. of the 10th ACM SIGKDD Inter. Conf. on Knowledge Disc. and Data Mining*, ser. KDD '04, 2004, pp. 99–108.

[20] I. Damgård, M. Geisler, and M. Krøigaard, "Efficient and secure comparison for on-line auctions," in *ACISP*, 2007, pp. 416–430.

[21] ——, "A correction to 'efficient and secure comparison for on-line auctions'," *IJACT*, vol. 1, no. 4, pp. 323–324, 2009.

[22] G. Danezis and E. De Cristofaro, "Fast and private genomic testing for disease susceptibility," in *Proc. of the 13th WPES*, 2014, pp. 31–34.

[23] Deloitte, "GCC eDiscovery survey, How ready are you?" https://tinyurl.com/jp2qwey, 2015.

[24] G. development team. (2016) GMP: The GNU multiple precision arithmetic library. [Online]. Available: https://gmplib.org/

[25] W. Dong, M. Charikar, and K. Li, "Asymmetric distance estimation with sketches for similarity search in high-dimensional spaces," in *SIGIR*, 2008, pp. 123–130.

[26] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft, "Privacy-preserving face recognition," in *Proc. of 9th PETS*, 2009, pp. 235–253.

[27] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. J. Strauss, and R. N. Wright, "Secure multiparty computation of approximations," *ACM Trans. on Algorithms*, vol. 2, no. 3, pp. 435–472, 2006.

[28] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *ACM CCS*, 2015, pp. 1322–1333.

[29] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart, "Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing," in *USENIX Security*, 2014, pp. 17–32.

[30] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *CRYPTO*, 2010, pp. 465–482.

[31] O. Goldreich, *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.

[32] ——, *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[33] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270 – 299, 1984.

[34] Google. (2017) Protocol buffers. [Online]. Available: http://code.google.com/apis/protocolbuffers/

[35] J. Hastad and A. Shamir, "The cryptographic security of truncated linearly related variables," in *STOC*, 1985, pp. 356–362.

[36] M. Henzinger, "Finding near-duplicate web pages: A large-scale evaluation of algorithms," in *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '06. ACM, 2006, pp. 284–291.

[37] Y. Ishai, J. Katz, E. Kushilevitz, Y. Lindell, and E. Petrank, "On achieving the "best of both worlds" in secure multiparty computation," *SIAM J. Comput.*, vol. 40, no. 1, pp. 122–141, 2011.

[38] S. Kamara, P. Mohassel, M. Raykova, and S. Sadeghian, "Scaling private set intersection to billion-element sets," in *FC*, 2014, pp. 195–215.

[39] S. Keelveedhi, M. Bellare, and T. Ristenpart, "Dupless: Server-aided encryption for deduplicated storage," in *USENIX*, 2013, pp. 179–194.

[40] A. Khedr, P. G. Gulak, and V. Vaikuntanathan, "SHIELD: scalable homomorphic implementation of encrypted data-classifiers," *IEEE Transactions on Computers*, vol. PP:99, 2015.

[41] R. L. Lagendijk, Z. Erkin, and M. Barni, "Encrypted signal processing for privacy protection: Conveying the utility of homomorphic encryption and multiparty computation," *IEEE Signal Process. Mag.*, vol. 30, no. 1, pp. 82–105, 2013.

[42] P. Li and A. C. König, "b-bit minwise hashing," in *WWW*, 2010, pp. 671–680.

[43] G. S. Manku, A. Jain, and A. Das Sarma, "Detecting near-duplicates for web crawling," in *WWW*, 2007, pp. 141–150.

[44] P. McDaniel, N. Papernot, and Z. B. Celik, "Machine learning in adversarial settings," *IEEE Security & Privacy*, vol. 14, no. 3, pp. 68–72, 2016.

[45] L. Melis, G. Danezis, and E. D. Cristofaro, "Efficient private statistics with succinct sketches," in *NDSS*, 2016.

[46] I. Mironov, M. Naor, and G. Segev, "Sketching in adversarial environments," *SIAM J. Comput.*, vol. 40, no. 6, pp. 1845–1870, 2011.

[47] M. Mitzenmacher, R. Pagh, and N. Pham, "Efficient estimation for high similarities using odd sketches," in *WWW*, 2014, pp. 109–118.

[48] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. New York, NY, USA: Cambridge University Press, 2005.

[49] M. Naor and E. Yogev, "Bloom filters in adversarial environments," in *Advances in Cryptology - CRYPTO'15*, 2015, pp. 565–584.

[50] M. Naveed, E. Ayday, E. W. Clayton, J. Fellay, C. A. Gunter, J.-P. Hubaux, B. A. Malin, and X. Wang, "Privacy in the genomic era," *ACM Comput. Surv.*, vol. 48, no. 1, pp. 6:1–6:44, Aug. 2015.

[51] D. Notterman, U. Alon, A. Sierk, and A. Levine, "Transcriptional gene expression profiles of colorectal adenoma, adenocarcinoma, and normal tissue examined by oligonucleotide arrays," *Cancer Research*, vol. 61, pp. 3124–3130, 2001.

[52] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT*, 1999, pp. 223–238.

[53] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *IEEE EuroS&P*, 2016.

[54] N. Papernot, P. D. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *IEEE S&P*, 2016.

[55] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *IEEE S&P*, 2013, pp. 238–252.

[56] A. Raghunathan, G. Segev, and S. P. Vadhan, "Deterministic public-key encryption for adaptively chosen plaintext distributions," in *EUROCRYPT*, 2013, pp. 93–110.

[57] A. Rajaraman and J. D. Ullman, *Mining of Massive Datasets*. New York, NY, USA: Cambridge University Press, 2011.

[58] A. Shrivastava and P. Li, "In defense of minhash over simhash," in *Proc. of AISTATS*, 2014, pp. 886–894.

[59] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *CoRR*, vol. abs/1312.6199, 2013.

[60] T. Urvoy, E. Chauveau, P. Filoche, and T. Lavergne, "Tracking web spam with html style similarities," *ACM Trans. Web*, vol. 2, no. 1, pp. 3:1–3:28, 2008.

[61] T. Veugen, "Encrypted integer division and secure comparison," *Int. Journal of Applied Cryptology*, vol. 3, no. 2, pp. 166–180, 2014.

[62] X. S. Wang, Y. Huang, Y. Zhao, H. Tang, X. Wang, and D. Bu, "Efficient genome-wide, privacy-preserving similar patient query based on private edit distance," in *ACM CCS*, 2015, pp. 492–503.

[63] D. J. Wu, T. Feng, M. Naehrig, and K. E. Lauter, "Privately evaluating decision trees and random forests," *PETS*, 2016.

8. Appendix

8.1. Perturbation Attack Proof

8.1.1. Proof of Theorem 4.1. If we augment set S with an element z_i picked uniformly at random from the set $[m]$ of available elements, we get $\Pr\left(\min\{\pi^{(j)}(S \cup \{z\})\} = \pi^{(j)}(z)\right) = 1/(|S|+1) = 1/(s+1)$ where $j \in [k]$ and $i \in [t]$. Therefore we sample t elements z_i in total and we test if the sample has smaller value than the current minimum, under the j-th permutation $\pi^{(j)}$. Let $Z_i^{(j)}$ be a random variable where $i \in [t]$ and $j \in [k]$. Random variable $Z_i^{(j)}$ takes value 1 if for the i-th sample $z_i \notin S$ holds $\pi^{(j)}(z_i) < \min\{\pi^{(j)}(S)\}$ and zero otherwise. The probability that $Z_i^{(j)}$ takes value 1 is:

$$\Pr(Z_i^{(j)} = 1) = \Pr(\min\{\pi^{(j)}(S \cup \{z_i\})\} = \pi^{(j)}(z_i)) = \frac{1}{s+1}$$

Let t be the number of distinct samples that we draw, then by $Z^{(j)}$ we denote the random variable such that $Z^{(j)} = \sum_{i=1}^{t} Z_i^{(j)}$ for the chosen t. Notice that since $\pi^{(j)}$ is a permutation if t samples from $[m]$ are distinct, then their corresponding outputs with respect to the permutation function are distinct; this is a part that we revisit in the proof. The probability that $Z^{(j)} = 0$ means that none of the sampled elements is smaller than the current minimum of S according to $\pi^{(j)}$. Given this probability we can compute the probability that at least one of the sampled elements elements is smaller than $\min\{\pi^{(j)}(S)\}$.

We use the following expression of the Chernoff bound [48]:

$$Pr(Y \leq (1-\delta)\mu) \leq e^{-\mu\delta^2/2}. \tag{6}$$

Let $t = 2c(s+1)\ln^2(s)$ be the number of distinct samples that we draw and define δ as $\delta = \frac{1}{\sqrt{\ln(s)}}$. Then the mean of $Z^{(j)}$ is $E[Z^{(j)}] = 2c(s+1)\ln^2(s)\frac{1}{(s+1)} = 2c\ln^2(s)$ and from (6) we have:

$$\Pr\left(Z^{(j)} \leq (1-\delta)\mu\right) = \Pr\left(Z^{(j)} \leq (1-\frac{1}{\sqrt{\ln(s)}})2c\ln^2(s)\right)$$

$$= \Pr\left(Z^{(j)} \leq \frac{\sqrt{\ln(s)}-1}{\sqrt{\ln(s)}}2c\sqrt{\ln(s)}\sqrt{\ln^3(s)}\right)$$

$$= \Pr(Z^{(j)} \leq 2c(\sqrt{\ln(s)}-1)\ln^{3/2}(s))$$

$$\leq e^{-(2c\ln^2(s))(\frac{1}{\sqrt{ln(s)}})^2\frac{1}{2}} = e^{-c\ln(s)} = \frac{1}{s^c}.$$

Also notice that for $s > 3$ we have $2c(\sqrt{\ln(s)} - 1)\ln^{3/2}(s) > 0$, therefore $\Pr(Z^{(j)} = 0) \leq \Pr(Z^{(j)} \leq 2c(\sqrt{\ln(s)}-1)\ln^{3/2}(s))$. Thus, we have $\Pr(Z^{(j)} \geq 1) = 1 - \Pr(Z^{(j)} = 0) \geq 1 - \frac{1}{s^c}$. The above event is based on the premise that we have $t = 2c(s+1)\ln^2(s)$ distinct elements z_i that are not in set S, which give t *distinct* outputs with respect to $\pi^{(j)}$. For efficiency reasons though, we use a hash function $h^{(j)}$ instead of a permutation $\pi^{(j)}$, thus the elements z_i for $i \in [t]$ do not necessarily give distinct outputs with respect to $h^{(j)}$.

Formally, let $h^{(j)} : [m] \to R$ be a hash function with domain $[m]$ and range R for which $|R| = n$. The attacker has to choose an input x_i from $[m]$ and as a second step compute $h^{(j)}(x_i) \in R$ in order to get an element from R.

This brings up the question, how many elements do we need to sample from $[m]$ in order to get $t = 2c(s+1)\ln^2(s)$ distinct elements from R that do not belong to set S? We answer the above question using a balls-and-bins argument on a given hash function $h^{(j)}$. The cardinality of R, that is $|R| = n$, represents the number of available bins. The number of samples that we draw, that is t', represents the number of balls that we throw. We define as $B_S^{(j)}$ the set of bins that contain a ball of the set S with respect to $h^{(j)}$. Let $t' \geq t$ be the number of balls that we throw, we compute the probability that t out of the total t' balls land in distinct bins and those bins do not belong to $B_S^{(j)}$.

Notice that the cardinality of $B_S^{(j)}$ can be at most s, therefore we compute the probability of landing t balls in distinct bins among the available $n - s$ bins. Let $X_q^{(j)}$ be the random variable that takes value 1 if the q-th ball lands in a bin that is empty where $q \in [t']$; takes value 0 otherwise. Then the probability that $X_q^{(j)}$ takes value 1 is $\Pr(X_q^{(j)} = 1) = \frac{n-s-(q-1)}{n} \geq \frac{n-s-(t'-1)}{n}$. For simplicity we approximate $\Pr(X_q^{(j)} = 1)$ with the value $\frac{n-s-(t'-1)}{n}$ which is a tight lower bound on the probability that the q-th ball lands in an empty bin.

Let $X^{(j)}$ be the random variable that counts how many balls landed in an empty bin, then $X^{(j)} = \sum_{q=1}^{t'} X_q^{(j)}$. Let t' take the value $t' = t\ln(s)$, then we have $X^{(j)} = \sum_{q=1}^{t\ln(s)} X_q^{(j)}$ and its expectation is:

$$E[X^{(j)}] = E[\sum_{q=1}^{t\ln(s)} X_q^{(j)}] = \sum_{q=1}^{t\ln(s)} E[X_q^{(j)}]$$

$$= t\ln(s)\frac{n-s-t\ln(s)}{n} = t\ln(s) - \frac{st\ln(s)}{n} - \frac{t^2\ln^2(s)}{n}.$$

Random variables $X_q^{(j)}$ for $q \in [t']$ are clearly dependent, therefore we can not use the Chernoff bound of equation (6). We continue the analysis using the Poisson approximation technique [48] so we can work with independent random variables instead. Thus we define random variable $Y_q^{(j)}$ with parameter $\frac{n-s-(t'-1)}{n}$ and form their sum as $Y^{(j)} = \sum_{q=1}^{t'} Y_q^{(j)}$ that has expectation $\mu = t\ln(s) - \frac{st\ln(s)}{n} - \frac{t^2\ln^2(s)}{n}$. Given the sum of t' Poisson random variables $Y^{(j)}$, we are interested in bounding the probability that more than t out of the t' have value 1. In case $t < \mu$ we can use the following Chernoff bound for the sum of independent Poisson random variables $Y^{(j)}$:

$$\Pr(Y^{(j)} \leq t) \leq \frac{e^{-\mu}(e\mu)^t}{t^t}$$

If inequality $t' < \mu$ holds then, the following lower bound on n should also hold:

$$t < \mu \Rightarrow t < t\ln(s) - \frac{st\ln(s)}{n} - \frac{t^2\ln^2(s)}{n}$$

$$\Rightarrow n > \frac{s\ln(s) + t\ln^2(s)}{\ln(s) - 1} > s + t\ln(s). \quad (7)$$

Thus assumption 1 is **(A1)** $n > s + t\ln(s)$. Using the above Chernoff bounds we have:

$$\Pr(Y^{(j)} \leq t) \leq \frac{e^{-\mu}(e\mu)^t}{t^t} = e^{-t\ln(s)+\frac{ts\ln(s)}{n}+\frac{t^2\ln^2(s)}{n}} e^t (\frac{\mu}{t})^t$$

$$= s^{-t} s^{\frac{ts}{n}} s^{\frac{t^2\ln(s)}{n}} s^{2c(s+1)\ln(s)} (\frac{\mu}{t})^t$$

$$= s^{-t} s^{\frac{ts}{n}} s^{\frac{t^2\ln(s)}{n}} s^{2c(s+1)\ln(s)} (t\ln(s) - \frac{st\ln(s)}{n} - \frac{t^2\ln^2(s)}{n})^t t^{-t}$$

$$\leq s^{-t} s^{\frac{ts}{n}} s^{\frac{t^2\ln(s)}{n}} s^{2c(s+1)\ln(s)} (\ln(s))^t$$

$$= (s^{\frac{s+t\ln(s)}{n}+\frac{2c(s+1)\ln(s)}{t}-1}\ln(s))^t.$$

We further assume that $n > (s + t\ln(s))\ln(s)$ which is stronger than **(A1)**, so we have

(A2) $n > (s+t\ln(s))\ln(s) \Rightarrow n > (s\ln(s)+2c(s+1)\ln^4(s))$

. We also use the fact that $t = 2c(s+1)\ln^2(s)$ in order to get the following expression:

$$\leq (s^{\frac{1}{\ln(n)}+\frac{1}{\ln(s)}-1}\ln(s))^t = (s^{\frac{1}{\ln(n)}+\frac{1}{\ln(s)}-1} s^{\log_s(\ln(s))})^t$$

$$= (s^{\frac{1}{\ln(n)}+\frac{1}{\ln(s)}-1} s^{\frac{\ln(\ln(s))}{\ln(s)}})^t = (s^{\frac{2+\ln(\ln(s))}{\ln(n)}-1})^t$$

$$= (s^{\frac{2+\ln(\ln(s))}{\ln(n)}2c(s+1)\ln^2(s) - 2c(s+1)\ln^2(s)})$$

$$= (s^{(2+\ln(\ln(s)))2c(s+1)\ln(s) - 2c(s+1)\ln^2(s)})$$

$$= (s^{-(2+\ln(\ln(s)))2c(s+1)\ln(s)+2c(s+1)\ln^2(s)})^{-1}$$

$$= \frac{1}{s^{2c(s+1)\ln(s)\left(\ln(s)-(2+\ln(\ln(s)))\right)}} \leq \frac{1}{s^{4c}}$$

, where we assumed that $(s+1)\ln(s)\big(\ln(s) - (2 + \ln(\ln(s)))\big) > 2$ which is true for $s > 24$.

Switching from the poisson approximation to the exact case (Corollary 5.9)

$$\Pr(Y^{(j)} \leq t) \leq \frac{1}{s^{4c}} \Rightarrow$$

$$\Pr(X^{(j)} \leq t) \leq e\sqrt{m}\frac{1}{s^{4c}} = \frac{e\sqrt{2c(s+1)\ln^3(s)}}{s^{4c}}$$

$$< \frac{e2c^{1/2}s^{1/2}\ln^{3/2}(s)}{s^{4c}} < \frac{6c^{1/2}}{s^{2c}}.$$

Thus we have that

$$\Pr(X^{(j)} = 0) \leq \Pr(X^{(j)} \leq t) \Rightarrow$$

$$\Pr(X^{(j)} \geq 1) = 1 - \Pr(X^{(j)} = 0) \geq 1 - \Pr(X^{(j)} \leq t) = 1 - \frac{6c^{1/2}}{s^{2c}}$$

, where we assume that $t \geq 0$.

We define as Q the random variable that takes value 1 if the output set S^+ of the algorithm builds a signature $\sigma(S^+)$ that is dissimilar to the signature $\sigma(S)$ of the input set and takes value 0 otherwise.

$$\Pr(Q = 1) \geq \Pr(Q = 1 | \bigcap_{1 \leq j \leq \kappa} X^{(j)} \geq 1) \Pr(\bigcap_{1 \leq j \leq \kappa} X^{(j)} \geq 1)$$

$$= \Pr(Q = 1 | \bigcap_{1 \leq j \leq \kappa} X^{(j)} \geq 1)(1 - \Pr(\bigcup_{1 \leq j \leq \kappa} X^{(j)} \geq 1))$$

, using the union bound we get:

$$\geq \Pr(Q = 1 | \bigcap_{1 \leq j \leq \kappa} X^{(j)} \geq 1)(1 - \sum_{j=1}^{\kappa} \Pr(X^{(j)} \geq 1))$$

$$\geq \Pr(Q = 1 | \bigcap_{1 \leq j \leq \kappa} X^{(j)} \geq 1)(1 - \kappa \cdot \frac{6c^{1/2}}{s^{2c}}) = 1 \cdot (1 - \frac{6\kappa c^{1/2}}{s^{2c}})$$

$$= 1 - \frac{6\kappa c^{1/2}}{s^{2c}}.$$

□

8.1.2. Proof Sketch of Theorem 4.2. The objective function is maximizing the similarity, or else minimizing the distance, between the perturbed vector $\vec{v^+} = \vec{v} + \vec{x}$ and $\vec{v}$. The final value of the objective function is denoted as ν and is passed as an output by the attack algorithm. The linear constraints capture the goal of the adversary to "flip" the bits of the cosine sketch $\sigma(\vec{v^+})$. Specifically, if $\text{sgn}(\vec{w_i}^T\vec{v})$ is 1 then bit σ_i is also 1 in the original sketch. Thus the inequality constraint

$$\text{sgn}(\vec{w_i}^T\vec{v}) \cdot (\vec{w_i}^T(\vec{v} + \vec{x})) \leq 0$$

is satisfied when $\vec{w_i}^T(\vec{v}+\vec{x})$ is less or equal to zero, which implies that σ_i will be 0 in the sketch of the perturbed vector $\vec{v} + \vec{x}$. Similarly, if $\text{sgn}(\vec{w_i}^T\vec{v})$ is 1 then $\vec{w_i}^T(\vec{v}+\vec{x})$ has to be greater or equal to zero in order to satisfy the inequality constraint, implying that bit σ_i will be 1 in the sketch of the perturbed vector. Thus, every solution within the feasible region flips all the bits of the sketch and achieves $\hat{d}_{cos}(\vec{v}, \vec{v^+}) = 1$. □

8.2. Overview of Presented Protocols

Overview of kIndHash. The client encrypts and sends the values $x, \ldots, x^{k-1}$ to the server. Using the homomorphic properties of the cryptosystem, the server computes $[\sum_{i=0}^{k-1} a_i x^i]$. Next, the server needs to apply a modulo p reduction which can not be performed homomorphically on the ciphertext $[\sum_{i=0}^{k-1} a_i x^i]$. Therefore, the server blinds the ciphertext $[\sum_{i=0}^{k-1} a_i x^i]$ with r and sends the result to the client. The client then decrypts and performs the modulo p operation on the blinded value, thus gaining no useful information about the evaluation of the polynomial. As a next step, the two parties privately compare the pair of values $(r \mod p)$ and $\big((r+\sum_{i=0}^{k-1} a_i x^i) \mod p\big)$. Based on the result of the comparison the server computes the output ciphertext $[\sum_{i=0}^{k-1} a_i x^i \mod p]$. Protocol kIndHash performs one private comparison of two $\log p$ bits integers (i.e., an invocation of $\mathcal{F}_{\text{PrvComp}}$), $3k+5$ homomorphic operations (e.g., exponentiation, multiplications, encryptions/decryptions, subtractions), and 3 rounds of communication.

Overview of UpdateOddSketch. As a first step protocol EncHashing is invoked, which returns to the server the hash value $h_{odd}(x) = (\sum_{i=0}^{k-1} a_i x^i) \mod u$ encrypted with the client's public key. As a next step we change the encryption from Paillier to DGK, since DGK embeds reductions modulo u to its homomorphic operations (see Section 2.4). Thus, the result of the blinding in step **(3)** is a random element from the range $[0, u-1]$. Then the server sends the blinded ciphertext $\langle h' \rangle$ to the client who decrypts it (step **(4)**) and prepares an encrypted bit mask for the server. The mask is a bit string of length u where every bit has value 0 except the bit in position h' which has value 1. We note here that the value of u is relatively small since it represents the length of the *succinct* odd sketch. Blinding the value h with r at step **(3)** has the effect of shifting the only "1" value of the bit mask by r positions. Therefore, as a final step, the server has to re-arrange the encrypted mask so as to remove the effect of the blinding with r. In order to cancel-out the effect of blinding, the server has to "pull" the 1 value back by r positions by applying the following transformation:

$$(|msk_0|, \ldots, |msk_{r-1}|, |msk_r|, \ldots, |msk_u|)$$
$$\rightsquigarrow (|msk_r|, \ldots, |msk_u|, |msk_0|, \ldots, |msk_{r-1}|).$$

The ciphertexts of the updated bit mask are multiplied with the GM ciphertexts of the input sketch, resulting in a homomorphic XOR operation—and, therefore, the desired bit is flipped.

A similar blinding process with Paillier, as opposed to DGK, would give an element r from the range $[0, 2^{\lambda+l}]$. For a standard instantiation $\lambda = 1024$, which gives an impractical number of ciphertexts in steps **(5)** and **(6)**, consequently making the protocol completely impractical. Protocol UpdateOddSketch invokes two protocols (EncHashing, ChangeEnc), performs $2u+3$ homomorphic operations, and one roundtrip.

Overview of FindMin. Initially the server assigns the first encrypted value as the current minimum $[min]$. Then we compare the current minimum with the next encrypted value using the protocol EncComp, which outputs the result of the comparison without revealing the encrypted values to the key holder (i.e., the client). Notice, however, that if the server iterates through the ciphertexts in the originally given order then the client can learn the index of the minimum value. To overcome this the server picks a random permutation π that is applied before any pairwise comparison (step **(1)**). Thus the client learns the index of the minimum value with respect to the secret random permutation that the server applied. After the execution of the comparison protocol the client returns a re-encryption $[c_i]$ of the smallest among the input values $[min], [y_{\pi(i)}]$, so as not to reveal to the server which of the two ciphertexts is smaller. Re-encryption (denoted as Refresh) can be achieved by either decrypting and re-encrypting the ciphertext, or by using the homomorphic properties of the cryptosystem to refresh the randomness. Since the client can decrypt $[min]$ and $[y_{\pi(i)}]$, the server blinds the ciphertexts using r_i and s_i so as to create the blinded ciphertexts $[b_i]$ and $[c_i]$. In the final step we deal with two cases. If the result of the comparison is $min < y_{\pi(i)}$ (i.e., $t_i = 1$) the server subtracts the blinding r_i from the value that the client returned. Otherwise the server subtracts s_i. Protocol FindMin performs $n-1$ encrypted comparisons of l bit integers, $8(n-1)$ homomorphic operations and $n-1$ roundtrips.

Overview of SketchingOdd. With the building blocks in place the sketching protocol is rather straight-forward. First, the server initializes the odd sketch of length u by encrypting it with the client's public key. Then the server computes the hash values using h_1^{min} by invoking kIndHash in step **(2)**, and finds the minimum hash value by invoking FindMin in step **(3)**. Next, the server updates the odd sketch by invoking protocol UpdateOddSketch, using the minhash value from the previous step as an input. The above process is repeated for all κ hash functions in order to compute the final sketch $|\sigma_1|, \ldots, |\sigma_\kappa|$ encrypted with the server's key. At the end of the protocol the final sketch is re-encrypted with the server's public key.

Overview of SketchingCosine. The client encrypts the value of each dimension of the input vector $\vec{v}$ using her Paillier public key and sends the ciphertexts to the server. The server uses the additive homomorphism of the cryptosystem in order to compute the encrypted inner product $\vec{v} \cdot \vec{w_i}$ in step **(2)**. Next, the server invokes the encrypted comparison protocol EncComparison2, in order to compare the encrypted inner product with the ciphertext of value 0, thus computing the sign of the inner product. If $d_1 < d_0$, which is equivalent to $\vec{w} \cdot \vec{v} < 0$, the output of the comparison protocol is 0 (which agrees with Equation (5)), otherwise the output of the comparison protocol is 1. We highlight the fact that the result of the comparison in step **(4)** is encrypted with the client's GM key. Due to the security properties of protocol EncComparison2, the client doesn't learn the result of the comparison—thus there is no need to randomly permute d_0 and d_1. The above process is repeated for all k vectors $\vec{w_i}$ in order to compute the final cosine sketch.

Protocol EncHashing

Client: $SK_P^{(C)}, k, p$		Server: $[x], \{a_i\}_{i=0}^{k-1}, p$
(2) $\forall i = 2, \ldots, k-1, h_i = D(SK_P^{(C)}, [h_i])$	$\xleftarrow{[h_2],\ldots,[h_{k-1}]}$	**(1)** $\forall i = 2, \ldots, k-1$, Pick $r_i \in (0, 2^l) \cap \mathbb{Z}$, $[h_i] \coloneqq [x]^{r_i} \mod N^2$
(3) $\forall i = 2, \ldots, k-1, [h'_i] \coloneqq E(PK_P^{(C)}, h_i^i)$	$\xrightarrow{[h'_2],\ldots,[h'_{k-1}]}$	**(4)** $\forall i = 2, \ldots, k-1, [x_i] \coloneqq [h'_i]^{r_i^{-i}} \mod N^2$
(6) $h' = D(SK_P^{(C)}, [h'])$ **(7)** $d = h' \mod p$	$\xleftarrow{[h']}$	**(5)** Pick $r \in \mathbb{Z}_u$, $[h'] \coloneqq [r] \cdot [a_0] \cdot [x]^{a_1} \prod_{i=2}^{k-1} [x_i^i]^{a_i} \mod N^2$ **(8)** $c = r \mod p$
	$\xleftrightarrow{\text{PrvComparison}(d,c)}$	Receive $[t]$ such that $t = 1$ if $d < c$
(9) $[d] \coloneqq E(PK_P^{(C)}, d)$	$\xrightarrow{[d]}$	**(10)** Output $[h] = [d] \cdot ([c])^{-1} \cdot [t]^p$

Protocol EncComparison

Client: $SK_P^{(S)}, SK_{GM}^{(S)}, l$		Server: $[a], [b], l$
		(1) $[x] \coloneqq [2^l] \cdot [b] \cdot [a]^{-1} \mod N^2$ **(2)** Pick a random $r \in (0, 2^l) \cap \mathbb{Z}$
(4) $z = D(SK_P^{(C)}, [z])$ **(5)** $d \coloneqq z \mod 2^l$	$\xleftarrow{[z]}$	**(3)** $[z] \coloneqq [x] \cdot [r] \mod N^2$
		(6) $c \coloneqq r \mod 2^l$
	$\xleftrightarrow{\text{PrvComparison}(d,c)}$	Receive $\lvert t' \rvert$ such that $t' = 1$ if $c < d$
(7) $\lvert z_l \rvert \leftarrow E(PK_{GM}^{(C)}, z_l)$	$\xrightarrow{\lvert z_l \rvert}$	
		(8) $\lvert r_l \rvert \coloneqq E(PK_{GM}^{(C)}, r_l)$
(10) Output $t = D(SK_{GM}^{(C)}, \lvert t \rvert)$	$\xleftarrow{\lvert t \rvert}$	**(9)** $\lvert t \rvert \coloneqq \lvert z_l \rvert \cdot \lvert r_l \rvert \cdot \lvert t' \rvert \cdot \lvert 1 \rvert$

Figure 9. Protocol EncComparison is a slight modification of the comparison protocol found in [10], [61]. Protocol EncComparison2 is the same up to step **(9)** where it terminates by outputting $|t|$ to the Server.

Overview of Reconstruct. Client C_A sends to client C_B the sketch σ^A that is encrypted with the server's GM key. As a next step, client C_B performs a homomorphic XOR operation between her encrypted sketch $|\sigma^B|$ and the received $|\sigma^A|$ in order to create $|\sigma'|$. Then client B applies a random permutation π to vector $|\sigma'|$ so as not to reveal to the server the positions that the two sketches match. Next, the server receives the permuted sketch and decrypts the κ ciphertexts using her secret key. The server counts the number of 1s among the decrypted bits and returns the result, denoted as c, to both clients. Finally, using the appropriate reconstruction equation (Equation (3) for Jaccard distance, and Equation (5) for cosine distance), the clients output the distance approximation.

8.3. Omitted Protocols

The omitted protocols are presented in Figure 9.

8.4. Security Proofs

Our security proofs take the classic simulation based approach for semi-honest adversaries on the hybrid model with ideal access to functions [13] and show that a party's view in a protocol execution is simulatable given its input, its output (if any), and access to a series of ideal functionalities. On the one hand we have the hybrid world were protocols have access to functions that are invoked by specific step of the protocol and on the other hand we have the ideal world where the simulator lives. Thus, the participating parties learn nothing from the protocol's execution beyond what can be derived from their input. For the sake of brevity we don't denote the public keys, whenever there is an encryption we indicate which public key is used.

Security of $\mathcal{F}_{\text{kIndHash}}$ (Lemma 5.1)

Lemma. *Protocol kIndHash correctly and securely computes $\mathcal{F}_{kIndHash}$ in the ($\mathcal{F}_{PrvComp}$)-hybrid model.*

Let's consider first the case where Client is corrupted, denoted as $\mathcal{A}$; notice that Client has no output. Thus we only need to show that a simulator can generate the view of incoming messages received by the $\mathcal{A}$.

Adversary (or simulator) $\mathcal{A}'$ is given $(SK_P^{(C)}, x, k, p)$ and 1^λ and works as follows:

- $\mathcal{A}'$ starts by simulating $\mathcal{A}$.
- $\mathcal{A}'$ receives $[x], \ldots, [x^{k-1}]$ from $\mathcal{A}$.
- $\mathcal{A}'$ picks a random $\widetilde{h'} \in (0, 2^{l+\lambda}) \cap \mathbb{Z}$, encrypts it to get $[h']$ using $PK_P^{(C)}$, and sends it to $\mathcal{A}$.
- $\mathcal{A}'$ receives $\widetilde{d}$ and $PK_P^{(C)}$ from $\mathcal{A}$ which are sent to $\mathcal{F}_{\text{PrvComp}}$.
- $\mathcal{A}'$ receives $[\widetilde{d}]$ from $\mathcal{A}$.
- $\mathcal{A}'$ outputs $\perp$.

Now we show that the view of $\mathcal{A}$ in the simulation with $\mathcal{A}'$ is indistinguishable from its view in a hybrid execution using a series of games.

- *Game-0:* Same as the hybrid execution.
- *Game-1:* Same as Game-0 except that h' in step **(3)** is replaced by with $\widetilde{h'} \in (0, 2^{l+\lambda}) \cap \mathbb{Z}$. In

Game-0 h' is the blinded value of $\prod_{i=1}^{k-1} \alpha_i x^i$ with random r, but in this game $\widetilde{h'}$ is picked uniformly at random from $(0, 2^{l+\lambda}) \cap \mathbb{Z}$. Thus the distribution of $h' = r + \prod_{i=1}^{k-1} \alpha_i x^i$ and $\widetilde{h'}$ are computational indistinguishable.

Thus, the view of $\mathcal{A}$ in the simulation with $\mathcal{A}'$ is indistinguishable from its view in a hybrid execution.

Let's consider the case where Server is corrupted. Simulator $\mathcal{A}'$ is given $(\{a_i\}_{i=0}^{k-1}, p)$, 1^λ, output $[h]$, and works as follows:

- $\mathcal{A}'$ starts by simulating $\mathcal{A}$.
- $\mathcal{A}'$ generates $k-1$ distinct encryptions of 1, namely $\{[\widetilde{x_i}]\}_{i=1}^{k-1}$, and sends them to $\mathcal{A}$.
- $\mathcal{A}'$ receives $[h']$.
- $\mathcal{A}'$ encrypts bit $\widetilde{t} = 1$ with $PK_P^{(C)}$ and sends $[\widetilde{t}]$ to $\mathcal{A}$.
- $\mathcal{A}'$ receives $\widetilde{c}$ from $\mathcal{A}$ which is sent to $\mathcal{F}_{\text{PrvComp}}$.
- $\mathcal{A}'$ encrypts the bit $[\widetilde{d}] = 1$ with $PK_P^{(C)}$ and sends it to $\mathcal{A}$.
- Finally $\mathcal{A}'$ outputs $[h]$.

The games for the security proof are the following:

- *Game-0:* Same as the hybrid execution.
- *Game-1:* Same as Game-0 except that the messages $x, \ldots, x^{k-1}$ in step (1) are replaced with $\{\widetilde{x_i}\}_{i=1}^{k-1}$ where all of them have value 1. By semantic security of the Paillier's cryptosystem the distribution of the ciphertexts $\{[x^i]\}_{i=1}^{k-1}$ and $\{[\widetilde{x_i}]\}_{i=1}^{k-1}$ are computationally indistinguishable.
- *Game-2:* Same as Game-1 except the encrypted bit $[t]$ returned by $\mathcal{F}_{\text{PrvComp}}$ in step **(7)** is replaced by the encryption of $\widetilde{t} = 1$. Since Server receives the ciphertext of Paillier, the messages $[t]$ and $[\widetilde{t}]$ are computationally indistinguishable.
- *Game-3:* Same as Game-2 except the encrypted value $[d]$ sent to the Server in step **(8)** is replaced by the encryption of $\widetilde{d} = 1$. As before, due to CPA security of Paillier the messages $[d]$ and $[\widetilde{d}]$ are computationally indistinguishable.

Thus, the view of $\mathcal{A}$ in the simulation with $\mathcal{A}'$ is indistinguishable from its view in a hybrid execution.

Security of $\mathcal{F}_{\text{FindMin}}$ (Lemma 5.3)

Lemma. *Protocol FindMin correctly and securely computes $\mathcal{F}_{FindMin}$ in the ($\mathcal{F}_{EncComp}$)-hybrid model.*

Let's assume that $\mathcal{A}$ corrupts the Client. Adversary (or simulator) $\mathcal{A}'$ is given $((PK_P^{(C)}, SK_P^{(C)}, PK_{GM}^{(C)}, SK_{GM}^{(C)}, l)$ and 1^λ and works as follows:

- $\mathcal{A}'$ starts by simulating $\mathcal{A}$.
- $\mathcal{A}'$ receives the input of $\mathcal{A}$ to $\mathcal{F}_{\text{EncComp}}$.
- $\mathcal{A}'$ sends to the $\mathcal{A}$ a random bit $\widetilde{t_i}$.
- $\mathcal{A}'$ picks a pair of random values $\widetilde{b_i}$ and $\widetilde{c_i}$ from the range $(0, 2^{l+\lambda}) \cap \mathbb{Z}$ and sends their encryption with $PK_P^{(C)}$ to $\mathcal{A}$.
- $\mathcal{A}'$ receives the ciphertexts $[c_i]$ and $[t_i]$.
- $\mathcal{A}'$ repeats the above four steps $n-1$ times.
- $\mathcal{A}'$ outputs $\perp$.

The games for the security proof are the following:

- *Game-0:* Same as the hybrid execution.
- *Game-1:* Same as Game-0 except that the output bit of the ideal function $\mathcal{F}_{\text{EncComp}}$ t_i in step **(3)** is replaced with the random bit $\widetilde{t_i}$. From the security of $\mathcal{F}_{\text{EncComp}}$ the client does not learn any information about the values that the Server compares. Since there is no prior information about the result of the comparison and no inferred information from ideal function, from Client's perspective any output of $\mathcal{F}_{\text{EncComp}}$ is equally probable.
- *Game-2:* Same as Game-1 except the values b_i and s_i in step **(4)** are replaced with a pair of random values $\widetilde{b_i}$ and $\widetilde{c_i}$ from the range $(0, 2^{l+\lambda}) \cap \mathbb{Z}$. In Game-1 b_i is the blinded value of min with random r_i, but in this game $\widetilde{b_i}$ is picked uniformly at random from $0, 2^{l+\lambda}) \cap \mathbb{Z}$. Thus the distribution of b_i and $\widetilde{b_i}$ is computationally indistinguishable. Similarly in Game-1, c_i is the blinded value of $y_{\pi(i)}$ with random s_i, but in this game $\widetilde{c_i}$ is picked uniformly at random from $(0, 2^{l+\lambda}) \cap \mathbb{Z}$. Thus the distribution of c_i and $\widetilde{c_i}$ is computationally indistinguishable.

Thus, the view of $\mathcal{A}$ in the simulation with $\mathcal{A}'$ is indistinguishable from its view in a hybrid execution.

Now let's assume that $\mathcal{A}$ corrupts the Server. Simulator $\mathcal{A}'$ is given $(\{[y_i]\}_{i=1}^n, l)$, 1^λ, the output $[min]$, and works as follows:

- $\mathcal{A}'$ starts by simulating $\mathcal{A}$.
- $\mathcal{A}'$ receives the input of $\mathcal{A}$ to function $\mathcal{F}_{\text{EncComp}}$.
- $\mathcal{A}'$ receives $[b_i]$ and $[c_i]$.
- $\mathcal{A}'$ picks a pair of random values $\widetilde{c_i}$ and $\widetilde{t_i}$ from the range $(0, 2^{l+\lambda}) \cap \mathbb{Z}$ and sends their encryption with $PK_P^{(C)}$ to $\mathcal{A}$.
- $\mathcal{A}'$ repeats the above three steps $n-1$ times.
- $\mathcal{A}'$ outputs $[min]$.

The games for the security proof are the following:

- *Game-0:* Same as the hybrid execution.
- *Game-1:* Same as Game-0 except that the transferred ciphertexts c_i and t_i before step **(7)** are replaced with the encryption of the random values $\widetilde{c_i}$ and $\widetilde{t_i}$ with the key $PK_P^{(C)}$. From the CPA security of the Paillier cryptosystem the distribution of $[c_i]$ and $[\widetilde{c_i}]$ is computationally indistinguishable. A similar argument holds for the distribution of $[t_i]$ and $[\widetilde{t_i}]$.

Security of $\mathcal{F}_{\text{UpdateOddSketch}}$ (Lemma 5.2)

Lemma. *Protocol UpdateOddSketch correctly and securely computes $\mathcal{F}_{UpdateOddSketch}$ in the ($\mathcal{F}_{EncHashing}$,$\mathcal{F}_{ChangeEnc}$)-hybrid model.*

Let's assume that $\mathcal{A}$ corrupts the Client. Adversary $\mathcal{A}'$ is given

$$(PK_{GM}^{(C)}, SK_{GM}^{(C)}, PK_P^{(C)}, SK_P^{(C)}, PK_{DGK}^{(C)}, SK_{DGK}^{(C)}, u, k)$$

,and 1^λ and works as follows:

- $\mathcal{A}'$ starts by simulating $\mathcal{A}$.
- $\mathcal{A}'$ receives the input of $\mathcal{A}$ to function $\mathcal{F}_{\text{EncHashing}}$.
- $\mathcal{A}'$ receives the input of $\mathcal{A}$ to function $\mathcal{F}_{\text{ChangeEnc}}$.
- $\mathcal{A}'$ picks a random value $\widetilde{h'}$ from space $\mathbb{Z}_u$ and sends its DGK encryption $\langle \widetilde{h'} \rangle$ with key $PK_{DGK}^{(C)}$ to $\mathcal{A}$.
- $\mathcal{A}'$ receives $|msk_0|, \ldots, |msk_{u-1}|$ from $\mathcal{A}$.
- $\mathcal{A}'$ outputs $\perp$.

The games for the security proof are the following:

- *Game-0:* Same as the hybrid execution.
- *Game-1:* Same as Game-0 except that h' of step **(3)** is replaced with a random value $\widetilde{h'}$ from space $\mathbb{Z}_u$. In Game-1 h' is the blinded value of h with random r, but in this game $\widetilde{h'}$ is picked uniformly at random from $\mathbb{Z}_u$. Recall that in the DGK cryptosystem the homomorphic operations are performed modulo u in the plaintext space. Thus the distribution of h' and $\widetilde{h'}$ is identical.

Let's assume that $\mathcal{A}$ corrupts the Server. Simulator $\mathcal{A}'$ is given
$(PK_{GM}^{(C)}, PK_P^{(C)}, PK_{DGK}^{(C)}, [x], \{\alpha_i\}_{i=0}^{k-1}, u, (|skt_0|, \ldots, |skt_{u-1}|))$
,the output $(|skt'_0|, \ldots, |skt'_{u-1}|)$, and 1^λ and works as follows:

- $\mathcal{A}'$ starts by simulating $\mathcal{A}$.
- $\mathcal{A}'$ receives the input of $\mathcal{A}$ to function $\mathcal{F}_{\text{EncHashing}}$.
- $\mathcal{A}'$ receives the input of $\mathcal{A}$ to function $\mathcal{F}_{\text{ChangeEnc}}$.
- $\mathcal{A}'$ receives $\langle h' \rangle$ which is encrypted with $PK_{DGK}^{(C)}$ from $\mathcal{A}$.
- $\mathcal{A}'$ picks uniformly at random u random bits $\widetilde{msk_0}, \ldots, \widetilde{msk_{u-1}}$. Then $\mathcal{A}'$ encrypts them with key $PK_{GM}^{(C)}$ and sends $|\widetilde{msk_0}|, \ldots, |\widetilde{msk_{u-1}}|$ to $\mathcal{A}$.
- $\mathcal{A}'$ outputs $(|skt'_0|, \ldots, |skt'_{u-1}|)$.

The games for the security proof are the following:

- *Game-0:* Same as the hybrid execution.
- *Game-1:* Same as Game-0 except that the bits $msk_0, \ldots, msk_{u-1}$ of step **(5)** are replaced with a random random bits $\widetilde{msk_0}, \ldots, \widetilde{msk_{u-1}}$. Notice that server gets to see bits $\widetilde{msk_0}, \ldots, \widetilde{msk_{u-1}}$ encrypted using GM. From the CPA security of the GM cryptosystem the distribution of $|\widetilde{msk_i}|$ and $|msk_i|$ is computationally indistinguishable, for all $i \in \{0, \ldots, u-1\}$.