A Formal Treatment of Homomorphic Encryption Based Outsourced Computation in the Universal Composability Framework

Wasilij Beskorovajnov[3], Sarai Eilebrecht[3], Yufan Jiang[1,2], and Jörn Müller-Quade[1,2,3]

[1] Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
[2] KASTEL Security Research Labs, Karlsruhe, Germany
{yufan.jiang, mueller-quade}@kit.edu
[3] FZI Research Center for Information Technology, Karlsruhe, Germany
{beskorovajnov, eilebrecht}@fzi.de

Abstract. The adoption of Homomorphic Encryption (HE) and Secure Function Evaluation (SFE) applications in the real world remains limited, even nearly 50 years after the introduction of HE. This is particularly unfortunate given the strong privacy and confidentiality guarantees these tools can offer to modern digital life.

While attempting to incorporate a simple straw-man PSI protocol into a web service for matching individuals based on their profiles, we encountered several shortcomings in current outsourcing frameworks. Existing outsourced protocols either require clients to perform tasks beyond merely contributing their inputs or rely on a non-collusion assumption between a server and a client, which appears implausible in standard web service scenarios.

To address these issues, we present, to the best of our knowledge, the first general construction for non-interactive outsourced computation based on black-box homomorphic encryption. This approach relies on a non-collusion assumption between two dedicated servers, which we consider more realistic in a web-service setting. Furthermore, we provide a proof of our construction within the Universal Composability (UC) framework, assuming semi-honest (i.e., passive) adversaries.

Unlike general one-sided two-party SFE protocols, our construction additionally requires sender privacy. Specifically, the sender must contribute its inputs solely in encrypted form. This ensures stronger privacy guarantees and broadens the applicability of the protocol.

Overall, the range of applications for our construction includes all one-sided two-party sender-private SFE protocols as well as server-based arithmetic computations on encrypted inputs. Finally, we demonstrate the practical applicability of our general outsourced computation framework by applying it to the specific use case of Outsourced Private Set Intersection (OPSI) in a real-world scenario, accompanied by a detailed evaluation of its efficiency.

Keywords: Homomorphic Encryption (HE) · Secure Function Evaluation (SFE) · Universal Composability (UC) · Outsourced Computation

1 Introduction

Since the groundbreaking work of Rivest et al. [48], the concept of homomorphic encryption (HE) has garnered significant attention from the scientific community. Despite being relatively straightforward in concept, HE—a specific type of public key encryption (PKE)— and its relatively low security requirements, offering IND-CPA or the more recent IND-CPA$^{\mathrm{D}}$ security within the well-established honest-but-curious (passive adversary) model has seen limited adoption in industry. So far, the technology has not been integrated into widespread practical use.

A similar issue is evident with multi-party computation (MPC). MPC applications are highly varied, ranging from secure statistical evaluation and data mining without exposing personal information [54], to confidential auctions without the need for third-party trust [12], and even training neural networks without revealing input data [37]. However, despite these diverse applications, MPC has not yet achieved mainstream adoption.

During our research on a specific variant of outsourced Private Set Intersection (PSI), namely outsourced balanced structure-aware PSI, we used inductive reasoning to identify several obstacles related to outsourced computation based on HE and MPC in general. Structure-aware PSI was initially introduced in [28], where only Alice's input was assumed to have a well-known structure, while Bob's input remained unstructured and potentially much larger. However, such an imbalance of inputs is rarely encountered in scenarios where human or product profiles need to be matched based on a list of private criteria with a well-defined structure. When considering balanced inputs with a well-known structure, the PSI protocol effectively reduces to a naive straw-man PSI protocol. While simple, it serves as an excellent candidate for a cryptographic protocol that can be easily explained to a broader audience, facilitating the adoption process. Additionally, this protocol suffers from communication bandwidth growing linearly with the input size. However, if the input size is small and the number of clients is expected to be limited, this may not be a significant issue. The more pressing concern is that this protocol remains interactive, requiring both parties to actively participate in each matching request. Outsourced computation offers a promising approach to eliminate the interactivity of such protocols.

Significant research [34,39,58,46,40,44,43,56,32,57,51] has been conducted in recent years to enhance outsourcing frameworks, making them secure and applicable across a wide range of scenarios. Overall, these frameworks fit into three distinct paradigms: (A) Secret Sharing of the secret key or Multi-Key Homomorphic Encryption (MHE) [39,40,46,58,56,44,43], (B) the Server-Client Non-Collusion Assumption, [34], and (C) outsourced multiparty computation where clients secret share their inputs to the working servers [32,51,57]. During our research on outsourcing a balanced structure-aware PSI protocol, we found all paradigms to be surprisingly unsuitable. The primary obstacle arises in a web service scenario where we cannot guarantee that all, or even a fraction of, clients who have outsourced their inputs will ever come online again. This limitation rules out paradigm (A), as it requires clients to actively participate in certain

stages of the protocol execution after the outsourcing phase. One could argue that a threshold scheme might reduce this requirement to a small fraction of clients. However, consider a web service like an online survey where the clients either secret-share the secret key of the HE or use an MHE. Even in such a setting, some clients might submit their responses once and never return. In such a setting, even a threshold scheme becomes impractical.

Paradigm (C) on the other hand is based on multiparty computation where the workers perform a MPC protocol on the clients' secret-shared and outsourced inputs. In this case, the clients only have to secret share their inputs to the workers and are able to go offline until the output phase. Another advantage is that clients do not have to be online simultaneously in order to retrieve the results. However, this approach is still not practical for our use case since the output clients have to be online while the working servers send their secret-shared results to the clients in order to get the output clients' individual results.

Paradigm (B) avoids this functional constraint but introduces another critical issue: the server-client non-collusion assumption. In a web service scenario, this assumption is unplausible because the usage threshold must be as low as possible. Consequently, it becomes almost inevitable for a server administrator to create their own client, thereby gaining access to the internal state of the client, which compromises the protocol's security. An alternative approach is to adopt a non-collusion assumption that is more plausible in a web service scenario. Yasuda [61] demonstrated that a non-collusion assumption between well-known servers can enable an architecture for biometric authentication that aligns well with web service scenarios like ours. Other notable examples of works explicitly relying on a server-server non-collusion assumption include the Information-Theoretic Private Information Retrieval (IT-PIR) protocol by Chor et al. [23], the contact tracing protocol ConTra Corona by Beskorovajnov et al. [10], and the multi-cloud storage system by Stefanov et al. [52].

In this work, we take a step toward establishing a third paradigm for outsourced computation. This paradigm is designed for protocols that leverage Homomorphic Encryption (HE) and Secure Function Evaluation (SFE) to build secure and efficient applications in web service scenarios based on a non-collusion assumption between servers. Our main contribution is a transformation of a two-round two-party secure function evaluation (SFE) protocol into an outsourced computation protocol. The given SFE protocol is based on homomorphic encryption with the additional requirement that the sender also has to encrypt its inputs. To demonstrate the security of our approach, we provide a Universal Composability (UC) proof that extends to the instantiated outsourced protocol. Additionally, we offer simple criteria that enable concise security analyses, allowing researchers to demonstrate that the instantiated protocol is secure within the UC framework without requiring a full security proof for the overall construction.

Contribution To the best of our knowledge, we are the first who give a general construction for non-interactive outsourced computation based on black-box homomorphic encryption. Our main design goal is both reusability of the same setup (i. e. the key pair of the HE) for several rounds and non-interactiveness of

the clients after outsourcing their inputs. Additionally we prove our construction within the Universal Composabillity framework in the presence of semi-honest (i. e. passive) adversaries. The range of applications covers all one-sided sender-private SFE protocols and server based arithmetic computation on encrypted inputs. This set of applications, however, does not cover the ad-hoc computation of any arbitrary function for the same setup, i. e. key pair, thus the evaluation function cannot be changed after the outsourcing phase. We present our contribution in the following structure:

- In Section 4 we describe our generic outsourced computation protocol and the corresponding ideal functionality. Finally, we present the generic theorem for outsourcing any one-sided two-party sender-private SFE protocol and provide a brief proof sketch of the universal composability simulation. The full proof is available in Appendix A.
- In Section 5 we discuss the challenges of instantiating the generic protocol from the previous section and deploying it in real-world systems.
- In Section 6 we present a particular instantiation of our general framework, i. e. OPSI and the results of our case study. A separate subsection is dedicated to related work on OPSI, focusing on enhancing a tutoring service matching platform with privacy-preserving guarantees. Finally we present evaluation results of the implementation.

2 Related Work

The notion of outsourcing multi-party computation was established by Kamara et al. [34]. This was the first work where clients are able to compute a protocol with the help of a remote server whilst not having to secret share their given inputs to the servers or letting the clients do some intensive computation. They, however, only focused on outsourcing the computation to one single entity, and therefore assumed no collusion between clients and the computing server. Additionally, they used Yao's garbled circuits in order to compute a function which is less efficient than computation using a somewhat homomorphic encryption scheme. Subsequently, López-Alt et al. [39] proposed a modified construction using a key-homomorphic encryption scheme. In this setting, each client generates its own key pair, encrypts its input using their own public key and outsources their ciphertext to the untrusted server. Due to the key-homomorphic property of the encryption scheme, the server is able to compute on the ciphertexts produced by different public keys, resulting in a ciphertext which is decryptable by the product of the clients' secret keys. This therefore was the first work in which the clients did not have to do any computation on the inputs besides encrypting them and the whole computation could be relayed to the remote server. This construction was refined in [40] where they defined a new type of homomorphic encryption scheme, called multi-key HE.

Building on this, a new line of research was established, e. g. [58,46]. Yet, all works based on [40] rely on the fact that the clients have to bea simoultaniosly online during the output phase in order to decrypt the result together. This

reliance was ruled out in [46] where the one-server setting was extended to a two-server setting while keeping the idea that clients are able to encrypt their inputs using self-generated key pairs. In [46], Peter et al. gave a two-server construction using the BCP cryptosystem [15]. This cryptosystem is a homomorphic encryption scheme with the additional property to generate a master secret key, similar to an attribute based encryption scheme. Using this master secret key, which is produced in the setup phase in order to generate the public parameters for key generation, one is able to decrypt any ciphertext that is created with a public key generated by using the same public parameters. In [46], the master secret key is held by a semi-honest server S_2 that does not interact with the clients but rather with the other server S_1 (and therefore may be viewed as a trusted execution environment of S_1). In order to compute on the clients' inputs, the clients produce their own key pair using the public parameters provided by S_2 and outsource their encrypted inputs to S_1. S_1 then blinds each ciphertext and sends all blinded ciphertexts to S_2 to re-encrypt all ciphertexts under one single public key. This is done by decrypting each blinded ciphertext and encrypting it again under the product of all clients' ciphertexts. S_2 then sends the re-encrypted ciphertexts to S_1 in order to compute on the ciphertexts (which are now encrypted using the same public key). Lastly, S_1 blinds the computational encrypted result, sends it to S_2 in order to re-encrypt it under the clients' own public keys and relays the resulting ciphertexts back to S_1 in order to send the clients their encrypted computational result. This involves heavy computation and communication done by S_1 and S_2 and is therefore not very efficient. Another work based on a multi-key homomorphic encryption scheme is the multi-party framework by Mouchet et al. [44]. In this framework the parties are categorized as input parties who provide the encrypted inputs, evaluating parties who compute over the encrypted inputs and the receiving party who gets the encrypted result. Note that an overlap of the different parties' roles is allowed, e. g. an input party can also be an evaluating party and on the other hand the evaluation may also be done by a remote server. Briefly described, the protocol works as follows: the input parties generate their own key pair in order to encrypt their inputs under their own public key and outsource their encrypted inputs to the computing parties. Due to the key-homomorphic structure of the used multi-key homomorphic encryption scheme, the evaluating parties are able to compute the encrypted result using the encrypted inputs. In the last step, the result is sent back to the input parties who re-encrypt the result under the receiver's public key. The drawback of this framework is that for the re-encryption of the result, all input parties have to be online due to the internal secret key being additively shared among the input parties which isn't desirable in some application scenarios e. g. a platform for finding a possible learning partner based on an outsourced private section protocol, such as described Section 6. This problem was partially ruled out in [43] where a threshold multiparty homomorphic encryption scheme (TMHE) was defined such that only a threshold of t out of N input parties is needed to re-encrypt the result. [56] constructed a two-server solution based on [46] with the improvement that no server has to decrypt any ciphertext by using

proxy re-encryption and can be seen a bit similar to our work. Yet, all research works based on [40] are merely proven in the real-ideal paradigm.

A different line of research uses MPC in order to delegate the computation of the clients' inputs. In this line of research, the servers, there often called workers, compute in an MPC manner on the clients' inputs. Therefore those constructions often rely on secret-sharing the clients' inputs. One example of work within this line of research is [32] which proved the security of their construction within the UC framework. Another example is the Trinocchio framework [51] which is a multi-client verifiable computation protocol. This protocol however is only proven within the real-ideal paradigm for an honest majority of the servers and assumed no collusion between servers and clients. [57] propose a two-servers framework based on the SPDZ protocol [24]. In this protocol, the clients outsource their inputs via secret sharing to the two servers and the two servers compute the result based on the SPDZ protocol. Although this protocol is secure against a malicious adversary, the security is merely proven in the real/ideal paradigm and therefore is only sequentially composable.

3 Preliminaries

In this section, we recap the notations and definitions including frameworks needed throughout this work.

3.1 Basic Notation

Throughout this work, we denote sets in upper case (e. g. the set M) and elements of a given set in lower case (e. g. $m \in M$). Vectors are written in boldface (e. g. $\mathbf{v} \in M^n$) and its i-th coefficient is denoted as $\mathbf{v}[i]$. Also, we denote the security parameter as λ. We say a function $negl(\cdot)$ is negligible if for every positive value $c \in \mathbb{N}$ and all sufficiently large $\lambda \in \mathbb{N}$ it holds that $negl(\lambda) < \lambda^{-c}$. A distribution ensemble $X = \{X(\lambda, a)\}_{a \in \mathcal{D}, \lambda \in \mathbb{N}}$ is an infinite sequence of random variables indexed by $a \in \mathcal{D}$ for some domain $\mathcal{D}$ and $\lambda \in \mathbb{N}$.

Two distribution ensembles $X = \{X(\lambda, a)\}_{\lambda \in \mathbb{N}, a \in \mathcal{D}}$ and $Y = \{Y(\lambda, a)\}_{\lambda \in \mathbb{N}, a \in \mathcal{D}}$ are *computaionally insdistinguishable*, denoted by $X \overset{c}{\approx} Y$, if for every non-uniform polynomial-time algorithm D there exists a negligible function $negl(\cdot)$ such that for every n and every $a \in \mathcal{D}$

$$|Pr[D(X(\lambda, a)) = 1] - Pr[D(Y(\lambda, a)) = 1]| \leq negl(\lambda).$$

3.2 Homomorphic Encryption Schemes

Definition 1 *A homomorphic encryption scheme consists of a tuple of three algorithms* $(\mathsf{GEN}, \mathsf{ENC}, \mathsf{DEC})$

- *The key generation algorithm* $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{GEN}(1^\lambda)$ *takes the security parameter* λ *and produces a random key pair* $(\mathsf{pk}, \mathsf{sk})$*, where* sk *denoted the secret*

key for decrypting a ciphertext which is held private by the ciphertext receiver and pk *denotes the public key that is used to produce a ciphertext of a given message.*
- *The probabilistic encryption algorithm* $c \leftarrow \mathsf{ENC}(\mathsf{pk}, m)$ *takes a message* $m \in \mathcal{M}$ *from the message space* $\mathcal{M}$ *and a public key and produces a ciphertext* $c \in \mathcal{C}$ *from the ciphertext space* $\mathcal{C}$ *using some additional randomness.*
- *The evaluation algorithm* $c_o = (C, ek, c_1, \ldots, c_n)$ *takes a circuit* C*, and evaluation key* ek *and several ciphertexts* $c_1, \ldots, c_n$ *and evaluates the circuit over the given ciphertexts, resulting in an output ciphertext* c_o*.*
- *The deterministic decryption algorithm* $m = \mathsf{DEC}(\mathsf{sk}, c)$ *takes a ciphertext* $c \in \mathcal{C}$ *from the ciphertext space* $\mathcal{C}$ *and computes the message corresponding to* c*.*

We require the the two following properties:

- Correctness: For every message $m \in \mathcal{M}$ out of the message space and every key pair $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{GEN}(1^\lambda)$ we require that
$$C(m_1, \ldots, m_n) = \mathsf{DEC}(\mathsf{sk}, C(\mathsf{ENC}(\mathsf{pk}, m_1), \ldots, \mathsf{ENC}(\mathsf{pk}, m_n))$$
for a given circuit C.
- IND-CPA security [29]: Let $PKE = (\mathsf{GEN}, \mathsf{ENC}, \mathsf{DEC})$ be an encryption scheme. PKE is said to be indistinguishable under chosen plaintext attacks (IND-CPA) secure, if for every security parameter λ and every PPT adversary $(\mathcal{A}, \mathcal{B})$, its advantage as defined as follows is negligible in λ:
$$Adv_{CPA} = |Pr[(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk}) : \mathcal{B}(\mathsf{pk}, \mathsf{ENC}(\mathsf{pk}, m_0)) = 1]$$
$$-Pr[(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk}) : \mathcal{B}(\mathsf{pk}, \mathsf{ENC}(\mathsf{pk}, m_1)) = 1]| = negl(\lambda)$$

Definition 2 ($\mathbf{IND-CPA^D}$ Security [38]) *Let* $E = (\mathsf{GEN}, \mathsf{ENC}, \mathsf{DEC}, \mathsf{EVAL})$ *be a public-key homomorphic (possibly approximate) encryption scheme with plaintext space* $\mathcal{M}$ *and ciphertext space* $\mathcal{C}$*. We define an experiment* $\mathrm{Expr}_b^{indcpa^D}[\mathcal{A}]$*, parameterized by a bit* $b \in \{0, 1\}$ *and involving an efficient adversary* $\mathcal{A}$ *that is given access to the following oracles, sharing a common state* $S \in (\mathcal{M} \times \mathcal{M} \times \mathcal{C})^*$ *consisting of a sequence of message-message-ciphertext triplets:*

- *An encryption oracle* $\mathcal{O}_{\mathsf{ENC}}(m_0, m_1)$ *that, given a pair of plaintext messages* m_0, m_1*, computes* $c \leftarrow \mathsf{ENC}_{\mathsf{pk}}(m_b)$*, extends the state*
$$S := [S; (m_0, m_1, c)]$$
with one more triplet, and returns the ciphertext c *to the adversary.*
- *An evaluation oracle* $\mathcal{O}_{\mathsf{EVAL}}(g, J)$ *that, given a function* $g : \mathcal{M}^k \rightarrow \mathcal{M}$ *and a sequence of indices* $J = (j_1, \ldots, j_k) \in \{1, \ldots, |S|\}^k$*, computes the ciphertext* $c \leftarrow \mathsf{EVAL}_{\mathsf{pk}}(g, S[j_1].c, \ldots, S[j_k].c)$*, extends the state*
$$S := [S; (g(S[j_1].m_0, \ldots, S[j_k].m_0), g(S[j_1].m1, \ldots, S[j_k].m_1), c)]$$
with one more triplet, and returns the ciphertext c *to the adversary, Here and below* $|S|$ *denotes the number of triplets in the sequence* S*, and* $S[j].m_0$*,* $S[j].m_1$ *and* $S[j].c$ *denote the three components of the* j*-th elememt of* S*.*

– *A decryption oracle* $O_{\mathsf{DEC}}(j)$ *that, given an index* $j \leq |S|$*, checks whether* $S[j].m_0 = S[j].m_1$*, and if so, returns* $\mathsf{DEC}_{\mathsf{sk}}(S[j].c)$ *to the adversary. (If the check fails, a special error symbol* $\perp$ *is returned.)*

The experiment is defined as

$$\begin{aligned}\mathrm{Expr}_b^{indcpa^D}[\mathcal{A}](1^\kappa)\text{: } & (sk, pk, ek) \leftarrow \mathsf{GEN}(1^\kappa)\\ & S := []\\ & b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{ENC}},\mathcal{O}_{\mathsf{EVAL}},\mathcal{O}_{\mathsf{DEC}}}(1^\kappa, \mathsf{pk}, \mathsf{ek})\\ & \text{return(b}')\end{aligned}$$

The advantage of adversary $\mathcal{A}$ *against the* $\mathrm{IND-CPA}^D$ *security scheme is*

$$\mathrm{Adv}_{\mathrm{indcpa}^D}[\mathcal{A}](\kappa) = |\Pr\{\mathrm{Expr}_0^{\mathrm{indcpa}^D}[\mathcal{A}](1^\kappa) = 1\} - \Pr\{\mathrm{Expr}_1^{\mathrm{indcpa}^D}[\mathcal{A}](1^\kappa) = 1\}|,$$

where the probability is over the randomness of $\mathcal{A}$ *and the experiment. The scheme* $\mathcal{E}$ *is* $\mathrm{IND\text{-}CPA}^{\mathrm{D}}$*-secure if for any efficient (probabilistic polynomial time)* $\mathcal{A}$*, the advantage* $\mathrm{Adv}_{indcpa^D}[\mathcal{A}]$ *is negligible in* κ*.*

3.3 Real/Ideal Simulation Paradigm

Since we base our overall outsourced SFE construction on a standalone secure two-party protocol, we briefly recap its security notion here. Loosely speaking, the so-called standalone security is a security framework where the security of a protocol is proved by providing a simulated protocol transcript where the simulator constructing the transcript only gets the inputs of a subset of the parties (more detailed, it gets the inputs of the corrupted parties) and showing that the simulated protocol execution is computationally indistinguishable from a real-world protocol execution between the actual parties. Put more formally, we set up two different worlds, called the real world, where the actual protocol is computed by the parties and a subset of the parties is (either semi-honestly or maliciously) controlled by an adversary, and the ideal world, where the the same function is computed by an incorruptable Turing machine, called the ideal functionality interacting with a simulator which is simulating the real-world execution by getting the inputs of the adversarially controlled parties and the function's output and generating a protocol transcript (consisting of the simulated messages that the simulated parties exchanged during the protocol execution) that is supposed to be indistinguishable from the transcript of the real-world execution. In order to prove the stated indistinguishability of the transcripts, we have a polynomially bounded interactive Turing machine, called the distinguisher, which gets all parties' inputs, the output and the transcript (which we call the distinguisher's view) and decides whether the transcript is generated by either the real-world interaction between the computing parties or by the simulator interacting with the functionality. Stated as a definition, we require the following to hold:

Definition 3 *Let* π *be a n-party protocol on inputs* $\mathbf{x} = (x_1, \ldots, x_n) \in X^n$ *and* $\mathcal{F}$ *a functionality. We say that* π *standalone securely realizes* $\mathcal{F}$ *if for all polynomially bounded adversaries* $\mathcal{A}$ *a simulator* $\mathcal{S}$ *exists such that:*

$$IDEAL_{\mathcal{F},\mathcal{S}}(\lambda,\mathbf{x})_{\lambda\in\mathbb{N},\mathbf{x}\in X^n} \overset{c}{\approx} REAL_{\pi,\mathcal{A}}(\lambda,\mathbf{x})_{\lambda\in\mathbb{N},\mathbf{x}\in X^n},$$

where $REAL_{\pi,\mathcal{A}}(\lambda,\mathbf{x})_{\lambda\in\mathbb{N},\mathbf{x}\in X^n}$ *is* $\mathcal{A}$*'s view of the real-world protocol execution and* $IDEAL_{\mathcal{F},\mathcal{S}}(\lambda,\mathbf{x})_{\lambda\in\mathbb{N},\mathbf{x}\in X^n}$ *is* $\mathcal{S}$*'s view in the ideal world when interacting with* $\mathcal{F}$*.*

3.4 Universal Composability Framework

Since we want to prove the more restricted UC security of our general outsourced SFE protocol, we shortly recapture the UC framework by Canetti [16,17]. The UC framework additionally introduces an interactive Turing machine $\mathcal{Z}$, called the environment, which has to distinguish the real world execution from the ideal world simulation by actively communicating with the adversary during the function computation. Additionally, we define the environment's view with an adversary and a function as the input, the exchanged messages between the corrupted parties and adversary together with the outputs. This restricts the simulation-based proof since rewinding of the adversary is not possible in this case. For a formal statement, we recite the simulation paradigm of the UC framework by Canetti.

Definition 4 ([17]) *Let* π *be a* n*-party protocol on inputs* $\mathbf{x} = (x_1,\ldots,x_n) \in X^n$ *and* $\mathcal{F}$ *a functionality. We say that* π *UC-securely realizes* $\mathcal{F}$ *if for all polynomially bounded adversaries* $\mathcal{A}$ *a simulator* $\mathcal{S}$ *exists such that for every polynomially bounded environment* $\mathcal{Z}$*:*

$$IDEAL_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda,\mathbf{x})_{\lambda\in\mathbb{N},\mathbf{x}\in X^n} \overset{c}{\approx} REAL_{\pi,\mathcal{A},\mathcal{Z}}(\lambda,\mathbf{x})_{\lambda\in\mathbb{N},\mathbf{x}\in X^n},$$

where $REAL_{\pi,\mathcal{A},\mathcal{Z}}(\lambda,\mathbf{x})_{\lambda\in\mathbb{N},\mathbf{x}\in X^n}$ *is the environment's view when interacting with* $P_1,\ldots,P_n$ *and* $\mathcal{A}$ *and* $IDEAL_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda,\mathbf{x})_{\lambda\in\mathbb{N},\mathbf{x}\in X^n}$ *is* $\mathcal{Z}$*'s view when interacting with* $\mathcal{F}$ *and* $\mathcal{S}$*.*

Additionally, the Universal Composition framework has the property that any UC-secure protocol can be securely (concurrently) composed with any other UC-secure protocol. This is the universal composition theorem, which is rephrased here without proof.

Theorem 1 (UC composition theorem, see [17]) *Let* $\mathcal{F}$ *and* $\mathcal{G}$ *be two ideal functionalities. Further let* ρ *be a protocol that securely UC-realizes* $\mathcal{G}$ *and let* π *be a protocol that securely UC-realizes* $\mathcal{F}$*. Then the composed protocol* ρ^{π} *securely UC-realizes* $\mathcal{G}$*.*

We refer the proof to [17].

4 Formal Modeling of Outsourced Computation

In this section, we present the construction for non-interactive outsourced computation based on black-box homomorphic encryption. To this end we firstly define the according ideal functionality together with a formal description of a real protocol. Eventually, we show a concise proof sketch for the security within the UC framework. The full proof can be found in Appendix A.

4.1 Ideal Functionality

For our outsourced secure function evaluation framework, we define a functionality viewed in Figure 1 that matches the quintessential interactions of participating parties in real world case studies. This includes the ability of clients to register to the protocol after setting up the protocol and model the clients' actions as separate phases from the actual computation rounds. Furthermore, we enable the reusability of client's outsourced inputs across multiple computation rounds in order to avoid unnecessary communication between clients and servers.

Therefore we split the standard outsourced secure function evaluation functionality into several distinct phases:

Registration First, we integrate a registration phase to the functionality. This phase is designed for new clients to join a fresh computation round without setting up a new protocol instance. It reflects the client registration process in the real world scenario, e. g. on a web service platform clients can register themselves while the actual protocol instance is already running.

Outsourcing The next two phases are the outsourcing phases for initiators' and clients' inputs. Those are modeled as separate phases apart from the actual computation phase due to two main reasons: First, we want to model the functionality in such a way that clients don't have to stay online during the actual computation round. The second reason is that we want to model the reusability of a client's inputs for several computation rounds. This is not guaranteed if the clients outsource their inputs during the computation phase.

Computation In the computation phase, the ideal functionality $\mathcal{F}_{\mathsf{OutComp}}$ receives a trigger from the initiating party $\mathsf{P}_{\mathsf{Y,j}}$ and computes the function f over the outsourced inputs of all input clients $\mathsf{P}_{\mathsf{X,i}}$ and the initiator $\mathsf{P}_{\mathsf{Y,j}}$.

Output The initiator $\mathsf{P}_{\mathsf{Y,j}}$ may may trigger the functionality $\mathcal{F}_{\mathsf{OutComp}}$ at any time and query for the computation result, which will be delivered to the client if it is finished.

The reason for decoupling the computation and output process into two independent phases is the modeling of a client not always being online during the computation. This is especially beneficial if the servers have to do complex computations since the client does not have to be idle during the whole computation in order to retrieve the round's computational result. Also, it models the real-world web service scenario more accurately: In such an application, a client might send a computation request and go offline after the request is handled. Then for the result retrieval, the client would request the server when being back online to check if the computation was finished.

Functionality $\mathcal{F}_{\mathsf{OutComp}}$

Setup The functionality is parameterized with an (m+1)-ary function $f : X^{m+1} \to Y$, and empty lists $L_{P,X}$ and $L_{P,Y}$. The functionality interacts with n initiators $\mathsf{P_{Y,j}}$ for $j \in [n]$, m input parties $\mathsf{P_{X,i}}$ for $i \in [m]$, the calculator server $\mathsf{S_C}$, the decryptor server $\mathsf{S_D}$ and the simulator $\mathcal{S}$. Additionally, initialize an empty list $R_{X,Y} = \varnothing$ for registered clients

Registration of Client $\mathsf{P_{Y,j}}$ or $\mathsf{P_{X,i}}$: Whenever a client C sends a message $(\mathsf{register}, \mathsf{sid}, \mathsf{pid})$ check whether pid is already in the list $R_{X,Y}$ of registered parties. If not, send back *ok* and add pid to $R_{X,Y}$, else send $\perp$.
Outsourcing of $\mathsf{P_{X,i}}$'s input: Whenever receiving the message $(\mathsf{outsource}, \mathsf{sid}, x_i, \mathsf{P_{X,i}})$ from an input party $\mathsf{P_{X,i}}$, add the entry $(\mathsf{P_{X,i}}, x_i)$ to the list of outsourced input parties' entries $L_{P,X}$. If there is already an existing entry $(\mathsf{P_{X,i}}, {x_i}')$ for the given input, replace the existing input ${x_i}'$ with the new input x_i. Additionally send the message $(\mathsf{input}, \mathsf{P_{X,i}})$ to $\mathcal{S}$
Outsourcing of $\mathsf{P_{Y,j}}$'s input: Whenever receiving the message $(\mathsf{outsource}, \mathsf{sid}, y_j, \mathsf{P_{Y,j}})$ from an initiator client $\mathsf{P_{Y,j}}$, add the entry $(\mathsf{P_{Y,j}}, y_j)$ to the list of outsourced input parties' entries $L_{P,Y}$. If there is already an exisiting entry for the given input, replace the existing input with the new input. Send the message $(\mathsf{input}, \mathsf{P_{Y,j}})$ to $\mathcal{S}$
Protocol Computation:
Upon receiving $(\mathsf{start}, \mathsf{sid}, \mathsf{ssid})$ from an initiator $\mathsf{P_{Y,j}}$, send $(\mathsf{ssid}, \mathsf{P_{Y,j}})$ to notify $\mathcal{S}$. If $\mathcal{S}$ returns $(\mathsf{ssid}, \mathsf{P_{Y,j}})$, send a notification $(\mathsf{ssid}, \mathsf{P_{Y,j}})$ to $\mathsf{S_C}$ and $\mathsf{S_D}$.
Upon receiving $(\mathsf{ready}, \mathsf{ssid})$ from $\mathsf{S_C}$ and $\mathsf{S_D}$, retrieve all $(x_i, \mathsf{P_{X,i}})$ for $i \in [m]$ from $L_{P,X}$ and $(y_j, \mathsf{P_{Y,j}}) \in L_{P,Y}$:

- If some $(x_i, \mathsf{P_{X,i}})$ has not been stored yet, send $(\mathsf{output}, x_i, \mathsf{fail})$ to the initiator $\mathsf{P_{Y,j}}$, $\mathsf{S_C}$ and $\mathsf{S_D}$.
- Else, compute $z \leftarrow f(y_j, \{x_i\}_{i \in [m]})$ and store $(\mathsf{sid}, \mathsf{ssid}, z)$. If $\mathsf{S_D}$ and $\mathsf{P_{Y,j}}$ are corrupted, send (result, z) to $\mathcal{S}$, otherwise, send $(\mathsf{result}, \mathsf{ssid})$ to $\mathcal{S}$.

Protocol output:
Upon receiving $(\mathsf{output?}, \mathsf{sid}, \mathsf{ssid})$ from $\mathsf{P_{Y,j}}$, check whether some entry $(\mathsf{sid}, \mathsf{ssid}, z)$ is stored. If this is the case, forward the message to $\mathcal{S}$, else output $(\mathsf{sid}, \mathsf{ssid}, \perp)$ to $\mathsf{P_{Y,j}}$. When getting answer ok from $\mathcal{S}$, send $(\mathsf{output}, \mathsf{ssid}, z)$ to the initiator $\mathsf{P_{Y,j}}$ and if $\mathsf{P_{Y,j}}$ is corrupted additionally to $\mathcal{S}$; else send $(\mathsf{output}, \mathsf{ssid}, \perp)$ to the initiator $\mathsf{P_{Y,j}}$.

Fig. 1. Outsourced Computation Functionality $\mathcal{F}_{\mathsf{OutComp}}$.

Protocol Π_{OutComp}

Private inputs: Each initiator $\mathsf{P}_{\mathsf{Y,j}}$ has input y_j, where $\mathcal{P}_{\mathsf{Y}} = \{\mathsf{P}_{\mathsf{Y},0}, ..., \mathsf{P}_{\mathsf{Y},n-1}\}$. Each input party $\mathsf{P}_{\mathsf{X,i}} \in \mathcal{P}_{\mathsf{X}}$ has input x_i, where $\mathcal{P}_{\mathsf{X}} = \{\mathsf{P}_{\mathsf{X},0}, ..., \mathsf{P}_{\mathsf{X},m-1}\}$.
Public inputs: Public parameters, an arithmetic circuit f_1 and a stand-alone secure two-round one-sided two-party protocol Π_{2PC} realizing a function $f_2 : X \times Y \to Z$.
Outputs: At any round q, $\mathsf{P}_{\mathsf{Y,j}}$ outputs $z_q \leftarrow f_2(y_j, f_1(x_1, \cdots, x_m))$.

Initialization of servers S_{C} and S_{D}:
S_{D} generates a fresh random key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GEN}(1^\lambda)$ and invokes $\mathcal{F}_{\mathsf{KRK}}$ with $(\mathsf{register}, \mathsf{sid}, \mathsf{pk}, \mathsf{sk})$ in order to register its generated key pair. S_{C} invokes $\mathcal{F}_{\mathsf{KRK}}$ with $(\mathsf{retrieve}, \mathsf{sid}, \mathsf{S}_{\mathsf{D}})$ in order to receive S_{D}' public key pk.

Registration of Client $\mathsf{P}_{\mathsf{Y,j}}$ or $\mathsf{P}_{\mathsf{X,i}}$:
Whenever a new client wants to register itself, it invokes $\mathcal{F}_{\mathsf{KRK}}$ with $(\mathsf{retrieve}, \mathsf{sid}, \mathsf{S}_{\mathsf{D}})$ to get the public key pk of S_{D}.

Outsourcing of $\mathsf{P}_{\mathsf{X,i}}$'s input:
Whenever an input client $\mathsf{P}_{\mathsf{X,i}}$ wants to outsource its input x_i, it encrypts its input using S_{D}'s pk to compute $c_{x,i} = \mathsf{ENC}(\mathsf{pk}, x_i)$ and sends $(\mathsf{sid}, \mathsf{pid}, c_{x,i})$ to S_{C}.

Outsourcing of $\mathsf{P}_{\mathsf{Y,j}}$'s input:
Whenever an initiating client $\mathsf{P}_{\mathsf{Y,j}}$ wants to outsource its input y_j, it encrypts its input using S_{D}'s pk to compute $c_{y,j} = \mathsf{ENC}(\mathsf{pk}, y_j)$. Then it generates a random mask $r_j \leftarrow \mathcal{M}$ and encrypts it using S_{D}'s pk as $c_{r,j} = \mathsf{ENC}(\mathsf{pk}, r_j)$. Finally, it sends $(\mathsf{sid}, \mathsf{pid}, c_{y,j}, c_{r,j})$ to S_{C}.

Protocol computation:

1. Whenever S_{C} receives a message $(\mathsf{start}, \mathsf{sid}, \mathsf{ssid})$, it checks whether ssid is stored yet. If this is the case, it outputs $\perp$ to $\mathsf{P}_{\mathsf{Y,j}}$. Otherwise, it stores ssid and computes $c_a \leftarrow f_1(\{x_i\}_{i\in[m]})$. Afterwards, it follows the sender's instructions (i.e. step 2) of the two-party protocol Π_{2PC}: on the values $c_{y,j}$ and c_a in order to retrieve $c_{a'}$ and masks the computation result $c_{d'} = c_{a'} + c_{r,j}$
2. S_{C} sends $(\mathsf{sid}, \mathsf{ssid}, c_{d'})$ to S_{D}, which decrypts $d' \leftarrow \mathsf{DEC}(\mathsf{sk}, c_{d'})$ and stores $(\mathsf{sid}, \mathsf{ssid}, d')$.

Output:

1. Whenever $\mathsf{P}_{\mathsf{Y,j}}$ requests S_{D} $(\mathsf{output?}, \mathsf{sid}, \mathsf{ssid})$, S_{D} checks whether some $(\mathsf{sid}, \mathsf{ssid}, d')$ is stored for given $(\mathsf{sid}, \mathsf{ssid})$. If this is the case, it sends $(\mathsf{sid}, \mathsf{ssid}, d')$ to $\mathsf{P}_{\mathsf{Y,j}}$. Else S_{D} sends $(\mathsf{sid}, \mathsf{ssid}, \mathsf{NotFinished})$ to $\mathsf{P}_{\mathsf{Y,j}}$.
2. $\mathsf{P}_{\mathsf{Y,j}}$ unmasks d' by computing $d = d' - r$, follows step 3 in Π_{2PC} on value d and outputs z.

Fig. 2. Outsourced Computation Protocol Π_{OutComp}

4.2 Protocol

The idea of our general outsourced multi-party protocol is to compose an arithmetic circuit (noted as f_1) over the inputs of all input clients, which can be implemented by a somewhat homomorphic encryption scheme, with a general function (noted as f_2) that can be possibly invoked by a one-sided two-round standalone two-party sender-private SFE protocol Π_{2PC} applying homomorphic encryption (a general construction of the two-party sender-private SFE protocol can be seen in Figure 3). The overall function f is thus decomposed into $f := f_1 \circ f_2$, where f_1 is secretly computed by $\mathsf{S_C}$ only and f_2 during the execution of protocol Π_{2PC}. We discuss different variants of f_1, f_2 later in Section 5.2. The reason why the required two-party protocol is specifically an SFE but not a more general MPC protocol is due the fact that the computation of the underlying protocol should be executed within one single computation phase and cannot be split over several distinct phases, such as commitments.

Compared to the functionality $\mathcal{F}_{\mathsf{OutComp}}$, the protocol Π_{OutComp} shown in Figure 2 has an additional server setup phase for the decryption server $\mathsf{S_D}$ to generate the key pair and sharing the generated public key with the calculation server $\mathsf{S_C}$. In this phase, the decryptor server generates its own key pair and registers it to the hybrid functionality $\mathcal{F}_{\mathsf{KRK}}$, which is defined according to [11] and can be found in Appendix B, where the key pair is stored and the public key is distributed to the calculator and each new registered client.

In the client registration phase, each new client invokes $\mathcal{F}_{\mathsf{KRK}}$ to receive the decryptor's public key. In the outsourcing phase, an input client encrypts its input using the decryptor's public key $c_{x,j} = \mathsf{ENC}(\mathsf{pk}, x_j)$ and sends the ciphertext to the calculator. An initiator additionally generates a masking value by generating a one-time-pad r, also encrypts it $c_r = \mathsf{ENC}(\mathsf{pk}, r)$ and sends the encrypted masking value together with the encrypted input to the calculator.

Each computation round is invoked by an initiator who sends the message $(\mathsf{start}, \mathsf{sid}, \mathsf{ssid})$ to the calculator, where ssid is a fresh sub-session id. Then, the calculator evaluates the arithmetic circuit f_1 over the input clients' inputs $c_a = f_1(c_{x,1}, ..., c_{x,n})$ and follows the instructions of two-party protocol Π_{2PC} for the sender (i. e. step 2 in Figure 3) in order to compute the encrypted interim result, which is forwarded to the decryptor afterwards. The decryptor decrypts the encrypted interim result and stores it for the result retrieval invoked by the initiator.

In the output phase, the initiator sends the sub-session id ssid to the decryptor to query the result. The decryptor goes through the stored interim results and checks whether the computation labeled with the given *ssid* is already finished. If not, the decryptor outputs notfinished and the initiator has to talk to the decryptor at a different time point. Otherwise, the decryptor sends the decrypted (masked) interim result to the initiator, who removes the masking one-time pad and follows the last step of the two-party computation protocol (i. e. step 3 in Figure 3) to compute the final result.

Protocol Π_{2PC}

Private inputs: The sender party S has input $\mathbf{x}$ and auxiliary information pk for a homomorphic encryption scheme HE and the receiver party R has input $\mathbf{y}$ and has auxiliary information $(\mathsf{sk}, \mathsf{pk})$.
Public inputs: Public parameters and an IND-CPA secure somewhat homomorphic encryption scheme $\mathsf{HE} = (\mathsf{GEN}, \mathsf{ENC}, \mathsf{DEC})$ and a set of functions $f, f_S, f_R : Y \times X \to Z$ with $f = f_R(\mathbf{y}, f_S(\mathbf{y}, \mathbf{x}))$ computable by HE
Outputs: R outputs $\mathbf{z} = f(\mathbf{x}, \mathbf{y})$.

Protocol computation:

1. R encrypts its input $c_{\mathbf{y}} = \mathsf{ENC}(\mathsf{pk}, \mathbf{y})$ and sends $(\mathsf{pk}, c_{\mathbf{y}})$ to S.
2. When S receives a message $(\mathsf{pk}, c_{\mathbf{y}})$ from R, S encrypts its input vector $c_{\mathbf{x}} = \mathsf{ENC}(\mathsf{pk}, \mathbf{x})$ and computes $c_{\mathbf{s}} = f_S(c_{\mathbf{y}}, c_{\mathbf{x}})$. S then sends $c_{\mathbf{s}}$ to R.
3. When R receives the ciphertext $c_{\mathbf{s}}$, it decrypts $c_{\mathbf{s}}$ to $\mathbf{z}' = \mathsf{DEC}(\mathsf{sk}, c_{\mathbf{s}})$. Then it evaluates $c_{\mathbf{s}}$ to $\mathbf{z} = f_R(\mathbf{z}', \mathbf{y})$ and outputs $\mathbf{z}$ at the end.

Fig. 3. The standalone secure one-sided sender-private 2-party SFE protocol Π_{2PC}

Underlying Two-Party Protocol The underlying two-party protocol as shown in Figure 3 used in our outsourced protocol is a non-interactive one-sided two-party sender-private function evaluation over two inputs, which realizes a simple functionality described in Figure 4. The only primitive required within our general construction is an IND-CPA^{test}D secure somewhat homomorphic encryption scheme. The general protocol structure works as follows: as a global setup, the receiver (or client) has a key pair of a somewhat homomorphic encryption scheme, the sender (or server) has the respective public key of the HE scheme. The global setup for both parties consist of the public parameters of the homomorphic encryption scheme and a composition $f = f_R(\mathbf{y}, f_S(\mathbf{y}, \mathbf{x}))$ of the function f to be computed on. Both functions f_S and f_R should be computable by the homomorphic encryption scheme and represent the computation instructions of both parties: f_S is the arithmetic circuit over the encrypted inputs $c_{\mathbf{x}}$ and $c_{\mathbf{y}}$ computed by the sender S and f_R represents the postprocessing done by the receiver R over the decrypted interim result $\mathbf{z}'$ and its input $\mathbf{y}$ in order to compute the protocol's result z. Additionally, the inputs of both parties are vectors of the same size. In the first step, the receiver encrypts their input using the public key $c_{\mathbf{y}} = \mathsf{ENC}(\mathsf{pk}, \mathbf{y})$ and sends the encrypted message to the sender. The sender then encrypts their own input $c_{\mathbf{x}} = \mathsf{ENC}(\mathsf{pk}, \mathbf{x})$ and evaluates a given arithmetic circuit $c_{\mathbf{s}} = f_S(c_{\mathbf{y}}, c_{\mathbf{x}})$ over the two encrypted inputs and sends the interim result back to the receiver. The receiver is then able to decrypt the interim result $\mathbf{z}' = \mathsf{DEC}(\mathsf{sk}, c_{\mathbf{S}})$. Depending on the evaluated function, the receiver might have to do some post-processing over the interim result in order to get the function's result. In this case, the client computes a local function $\mathbf{z} = f_R(\mathbf{z}', \mathbf{y})$ over the interim result $\mathbf{z}'$ and its local input and outputs the given result as the protocol's result.

Functionality $\mathcal{F}_{\mathsf{2PC}}$

The functionality is parameterized by a function over two parameters $f : X \times Y \to Z$. The functionality interacts with two parties, the sender S and the receiver R.

execution

– When getting input set X from S and input set R from R, compute the result $z = f(x, y)$ and output z to R.

Fig. 4. Two-Party Functionality $\mathcal{F}_{\mathsf{2PC}}$.

Sender-Private Property This property stems from the sender contributing its inputs only in an encrypted form. In general, there are one-sided two-party SFE protocols which do not require the sender to encrypt their inputs, such as in [18]. However, we require this additional encryption step on the sender's side since in our outsourced protocol, the dataset stored in $\mathsf{S_C}$ is in encrypted form due to the privacy of the clients. If such a requirement is in place, which is usually the case for outsourced computation, then protocols such as [18] cannot be used as is. Every such non-interactive one-sided two-party sender-private secure function evaluation protocol can be used as an underlying protocol for our general outsourced construction, if we make a small modification: The key pair used for the homomorhic encryption scheme must be a global setup across multiple rounds (in a single session). This modification does not affect the standalone security of the underlying protocol, but is merely needed for the proper security reduction in Appendix A.

4.3 Security

In the following, we give a formal theorem of the security of the overall protocol and a short proof sketch. Since the whole communication of all parties is over ciphertexts using the somewhat homomorphic encryption scheme—except the communication between the decryptor and the initiator—the simulation is straightforward since the simulator is either able to extract the correct inputs (hence we have to require to be in the $\mathcal{F}_{\mathsf{KRK}}$-hybrid model) or to simulate a legitimate ciphertext message due to the IND-CPA$^{\mathrm{D}}$ security of the homomorphic encryption scheme. The communication between the decryptor and the initiator is also simulatable since the message is a legitimate (decrypted) message of the protocol Π_{2PC} that standalone securely realizes f_2 where initiator's encrypted input and the output of f_1 can be viewed as the the encrypted inputs of Π_{2PC}.

Having the composition of f using f_1 and f_2 in mind, we are able to show how one execution round of our protocol Π_{OutComp} can be transformed into a standalone one-sided two-party sender-private protocol Π_{2PC}: Merge the input clients $\mathsf{P_{X,i}}$ and the calculator $\mathsf{S_C}$ into one party $(\mathsf{P_{X,i}}, \mathsf{S_C})$ and view the output

encrypted of f_1 (which is a circuit evaluation solely over the input clients) as one encrypted input of f_2. Also merge the decryptor $\mathsf{S_D}$ and the initiator $\mathsf{P_{Y,j}}$ into one party $(\mathsf{S_C}, \mathsf{P_{Y,j}})$.

The result is the computation of f_2. Stated more formally, we have the following theorem:

Theorem 2 *Assume a one-sided two-round two-party sender-private SFE protocol Π_{2PC} standalone-securely realizing a two-party functionality f_2 (viewed in Figure 4 that bases solely on a somewhat homomorphic encryption scheme, an arithmetic circuit c_f realizing a function f_1 using an IND-CPA^D secure somewhat homomorphic encryption scheme $\mathsf{HE} = (\mathsf{GEN}, \mathsf{ENC}, \mathsf{DEC})$ and a function f in $\mathcal{F}_{\mathsf{OutComp}}$ is defined as $f = f_1 \circ f_2$. Then the protocol Π_{OutComp} realizes the functionality $\mathcal{F}_{\mathsf{OutComp}}$ in the $\mathcal{F}_{\mathsf{KRK}}$-hybrid model with static corruption in the presence of a semi-honest adversary, assuming that the servers $\mathsf{S_C}$ and $\mathsf{S_D}$ do not collude.*

Proofsketch Due to page limitations, we only give a rough intuition why the construction provides the stated security property:

For the whole proof-sketch, we assume the dummy-adversary, which only forwards corruption commands sent by the environment to the corrupted party and forwards any messages coming from the corrupted party to the environment.

Corrupted Initiator The communication between a corrupted initiator and other parties are: In the outsourcing phase with the calculator, where it sends its inputs to be outsourced; in the computation phase, where it sends a computation initialization message, but gets no other message and in the output phase, where it gets the interim result of the computation by solely communicating with the decryptor, which is the interim result of Π_{2PC} on the initiator's inputs and the circuit evaluation $f_1(x_1, \ldots, x_m)$ over the input clients' inputs. In the ideal world, the simulator is able to setup a key pair on its own by simulating $\mathcal{F}_{\mathsf{KRK}}$, extract the initiator's (modified) input using the self-generated secret key and compute the simulation of Π_{2PC} in order to compute a valid message coming from the decryptor. Note that even if multiple initiators are corrupted, they are not able to learn more than their inputs and the respective outputs, since each computation and output phase is computed sequentially, which does not conflict with the security of the underlying standalone secure two-party protocol Π_{2PC}.

Corrupted Input Client The interaction between a corrupted input client with the calculator server in the outsourcing phase is the only communication takes place, where the corrupted input client outsources its (potentially modified) inputs. In the ideal world, the simulator is therefore able to generate its own key pair by simulating the hybrid functionality $\mathcal{F}_{\mathsf{KRK}}$ and is able to extract the (modified) inputs sent by the environment. Note that even if several input clients are corrupted, the corrupted parties can not learn more than their given inputs.

Corrupted Initiator and Input Client In the case of a corrupted initiator (or multiple corrupted initiators) and a corrupted input client (or multiple input

clients), the corrupted parties still only communicate with the two servers and therefore are not able to learn the other parties' inputs other than the outputs provide. In the ideal world, the simulator is able to use a self-generated key pair of the HE scheme by simulating the hybrid functionality $\mathcal{F}_{\mathsf{KRK}}$ and thus able to extract the (modified) inputs of the corrupted client parties. By the standalone security of the underlying protocol Π_{2PC}, the simulator is able to follow Π_{2PC}'s simulation in order to generate an appropriate message sent by the decryptor to the corrupted initiator. Again, since the given phases are computed only sequentially, the security of the construction can be reduced to the standalone security of Π_{2PC}, even if multiple client parties are corrupted.

Corrupted Calculator, Initiators and Input Client In the case of a corrupted calculator, if additionally some initiators and/or input client are corrupted, the adversary is not able to gain more information than the corrupted clients' inputs and the outputs given to the initiators: Since the calculator does not hold the secret key for the HE scheme and is only given ciphertexts, the adversary can not learn more information than the ciphertexts offer by themselves and the inputs, interim results and outputs of the corrupted parties, even after computing f_1 and following the sender's part (i. e. step 2) of Π_{2PC}, see Figure 3. In the ideal world, the simulator is able to extract the corrupted parties' inputs by simulating $\mathcal{F}_{\mathsf{KRK}}$ and generating a key pair on its own on behalf of the simulated $\mathcal{F}_{\mathsf{KRK}}$. Since the corrupted calculator receives all encrypted inputs of all clients (including the honest ones), the simulator has to produce the encrypted inputs of the honest clients. Due to the IND-CPA$^{\mathrm{D}}$ security of the HE scheme, $\mathcal{S}$ can easily fake those messages by generating some random ciphertexts. If additionally some initiators are corrupted, the simulator also is able to generate a valid message sent from the decryptor to the initiator by following the simulation of Π_{2PC}.

Corrupted Decryptor, Initiators and Input Clients In the case of a corrupted decryptor, if additionally multiple initiators or input clients are corrupted, the adversary is not able to learn more than the inputs and outputs of the corrupted clients, since the clients only forward the inputs and the decrypter only receives the interim results of Π_{2PC} on the encrypted inputs of one initiator's input and f_1's evaluation result. Additionally, since the decryptor only gets some message consisting of a masked evaluation by the calculator, the ability to decrypt the message does not help the adversary gain more information than the encrypted counterpart. In the ideal world, the simulator is able to learn $\mathcal{Z}$'s secret key dedicated for the corrupted decrypter by simulating $\mathcal{F}_{\mathsf{KRK}}$ and therefore getting the whole key pair for HE. Since $\mathcal{S}$ then holds the secret key, it is able to extract the corrupted parties' inputs and due to the standalone security of the underlying protocol Π_{2PC}, $\mathcal{S}$ is able to follow Π_{2PC} simulation instructions to generate a valid message coming from the honest calculator and to all honest initiators in each computation and output phase. Since we assume that the decryptor does not collude with the calculator, the corrupted decryptor is not able to learn the honest clients' encrypted inputs.

The whole proof can be seen in Appendix A. □

5 Instantiations

In this section we will examine the selection of abstract protocol parameters, as well as some of the more important implicit and explicit assumptions of Π_{OutComp}, and their implications for security and privacy.

5.1 IND-CPA$^{\mathrm{D}}$ Attacks on HE based Outsourced Computation

The works [38,20] have shown that outsourced computation based on both approximate and exact lattice-based FHE schemes cannot, in general, be reduced to the IND-CPA$^{\mathrm{D}}$ security of these schemes.

IND-CPA^{D} Attacks The authors describe attacks on various implementations of CKKS [21], BGV/BFV [13,27,14], and TFHE [22] that result in full secret key recovery. The attacks in [38] target the differences in approximate evaluations, while those in [20] exploit the observation that the correctness of exact FHE schemes is tightly coupled with the noise distribution. This distribution can be exploited by repeatedly summing ciphertexts of zeros until the output flips to a 1. Simply put, this makes it possible to compute the exact magnitude of the noise and, subsequently, recover the secret key. To guarantee security in general, a strengthening of FHE security is therefore required, which the authors of [38] introduce as IND-CPA$^{\mathrm{D}}$ security. This definition can be achieved through various means, and determining which measure offers the best trade-off between security and efficiency is an ongoing debate. However, all proposed measures require control over the noise distribution during computations. A promising approach is presented by Alexandru et al. [6], who introduced application-aware homomorphic encryption. This approach suggests controlling noise based on the specific application being computed, rather than assuming worst-case noise consumption. This strategy appears particularly suitable for our outsourcing framework because the outsourced protocol is restricted to a predefined evaluation function, which cannot be altered for a specific key pair after its initial setup. Unfortunately, as discussed in [20], this approach introduces an additional challenge in the engineering of HE-based outsourced computation. Further research is needed to streamline this process and reduce its susceptibility to errors.

Mitigation Strategies for Π_{OutComp} We have identified that, although IND-CPA$^{\mathrm{D}}$ security is generally required, there are specific scenarios in our outsourcing framework where the attack requirements, as described in [38,20], are not met. Consequently, the attack can be thwarted through organizational measures, and IND-CPA$^{\mathrm{D}}$ security suffices. For exact schemes, the attack requirements correspond to the query types of the IND-CPA$^{\mathrm{D}}$ security game:

- (A) Decryption requests
- (B) Evaluation requests
- (C) Encryption requests

All three types of requests are required to successfully carry out the attack. In our outsourced computation framework, we have two dedicated parties: the Calculator ($\mathsf{S_C}$) and the Decryptor ($\mathsf{S_D}$), who are assumed not to collude with each other. Additional parties include the Initiating Party ($\mathsf{P_{Y,j}}$) and the Input Client Party ($\mathsf{P_{X,i}}$). These parties can collude either with the Calculator or the Decryptor, but never with both simultaneously. In section 6, we describe our outsourced PSI protocol, which plausibly satisfies all the attack requirements if an adversary successfully corrupts the parties $\mathsf{S_C}$, $\mathsf{P_{Y,j}}$, and $\mathsf{P_{X,i}}$. Therefore, only an IND-CPA$^{\mathrm{D}}$ secure HE scheme can provide protection in this case.

The first key requirement is that the Calculator must be corrupted, enabling the issuance of type (B) requests. This scenario is plausible in an honest-but-curious setting. The second requirement is that some input clients must collude with the honest-but-curious calculating server, allowing the adversary to issue encryption requests (C). This is also plausible, as previously discussed, since non-collusion between a server and a client cannot be reliably assumed in a web service scenario. The final requirement, however, can only be met if the adversary (controlling the Calculator and some input clients) also gains access to the decryption results, which are only visible to the Initiating Party. While such a scenario is plausible in protocols similar to our OPSI protocol described in section 6, it is not universally applicable.

One important category of outsourced protocols in our framework that does not meet all three requirements simultaneously are protocols where the secret key resides within the Initiating Party. In this case, the Decryptor ($\mathsf{S_D}$) and the Initiating Party ($\mathsf{P_{Y,j}}$) effectively become a single party, denoted as $(\mathsf{S_D}, \mathsf{P_{Y,j}})$. This simplifies the protocol but also increases the need for proper authorization and authentication measures, as $(\mathsf{S_D}, \mathsf{P_{Y,j}})$ essentially becomes just another client among many, which, however, is assumed not to collude with $\mathsf{S_C}$. Input clients ($\mathsf{P_{X,i}}$) must then be assured that $(\mathsf{S_D}, \mathsf{P_{Y,j}})$ is not a fake client controlled by $\mathsf{S_C}$.[4] [5] In this case, the masking procedure can be skipped. One example of such an application is an online survey or private analytics/aggregations, where the decryption key resides on the initiating party's laptop, which creates the survey/analysis and keeps the final results private.

5.2 Choice of the Outsourced Function f

At the heart of every instantiation of Π_{OutComp} is the choice of the function f that will be computed on the encrypted inputs of the input clients $\mathsf{P_{X,i}}$ and the initiating party $\mathsf{P_{Y,j}}$. In Π_{OutComp}, we model the function f as a concatenation of f_1 and f_2, i. e., $f := f_2(y_j, f_1(x_1, \cdots, x_m))$:

[4] If $\mathsf{S_D}$ is instead a dedicated server, this assurance becomes easier to establish, as $\mathsf{S_D}$ can act as a well-known global entity trusted by all input clients ($\mathsf{P_{X,i}}$).

[5] Additionally, there might be data and communication overhead if an input client's data is to be reused across multiple computations initiated by different initiators, as the same input would need to be encrypted multiple times under different $(\mathsf{S_D}, \mathsf{P_{Y,j}})$ public keys.

- f_1 is an arithmetic circuit that is efficiently computable using a somewhat homomorphic encryption scheme and is solely computed on the inputs from the input clients $\mathsf{P}_{\mathsf{X},\mathsf{i}}$
- f_2 is a function that we require to be standalone-securely realizable by a protocol Π_{2PC} from Figure 3 and therefore this function also incorporates the input from the initiating party $\mathsf{P}_{\mathsf{Y},\mathsf{j}}$.

This way we are able to reduce the security and privacy of the overall outsourced protocol in the UC framework to the security and privacy of a simpler protocol in the standalone setting. We observe multiple cases:

1. Outsourced Arithmetic Circuit Calculation (OACC) case, where $f_2 := id$
2. Outsourced Secure Function Evaluation (OSFE) case, where $f_2 \neq id$, i. e., $\Pi_{\mathsf{2PC}}(y, x) = f_2(y, x) \neq (y, x)$
   - Simple two-party sender-private SFE: Receiver does no post-processing after the decryption step 3 in Π_{2PC} from Figure 3, i. e., $\mathbf{z} = \mathbf{z}'$.
   - Proper two-party sender-private SFE: Receiver does non-trivial post-processing, i. e., $\mathbf{z} \neq \mathbf{z}'$

The OACC case effectively means that the initiating party, $\mathsf{P}_{\mathsf{Y},\mathsf{j}}$, has no input that needs to be incorporated into the computation. In this scenario, no additional proofs are required beyond the IND-CPA$^{\mathrm{D}}$ security of the employed homomorphic encryption, leading to a secure instantiation of Π_{OutComp}. The OSFE case covers all outsourced protocols where the input of the initiating party $\mathsf{P}_{\mathsf{Y},\mathsf{j}}$ must be integrated into the computation. In this context, we can make two important distinctions that may simplify the protocol instantiation process. If the only task for the the initiator $\mathsf{P}_{\mathsf{Y},\mathsf{j}}$ is to unmask or decrypt the result for the protocol to successfully terminate, i. e., $\Pi_{\mathsf{2PC}}(y, x) = f_2(y, x)$, then the simulation will succeed for the instantiation Π_{OutComp}. The most complex case arises when the receiver is required to perform non-trivial computations after the decryption/unmasking step. In this case, one must either employ a Π_{2PC} protocol that already has a standalone simulation-based proof in the literature, or provide this proof themselves. However, we argue that the current state-of-the-art in SFE protocols is rich in primitives such as Private Set Intersection, Oblivious Transfer, Private Information Retrieval, Private Equality Tests, and many others that can be used as building blocks for a secure instantiation of Π_{OutComp}. Later in this section, we will describe a case study involving a non-trivial PSI protocol.

Output Privacy The formulation of privacy requirements in SFE and MPC protocols is built around the principle that a protocol should not reveal more information than what can be inferred from the result alone. Any information about the inputs that can be inferred from the result is usually termed *leakage*. A key question that remains unresolved is how to quantify the privacy loss due to this leakage, and consequently, how to sanitize the result before publication to limit the leakage to a predetermined, controlled threshold while retaining a sufficient level of usefulness or utility in the final output. This problem seems to

be equivalent to the general challenge that has driven anonymization and privacy research for decades. While this research has yielded numerous solutions for specific datasets and use cases, a universal solution for privacy definition and a corresponding sanitization method applicable to any type of information remains elusive. Currently, Differential Privacy (DP) [26], along with methods like the exponential mechanism from [41], may appear to offer a potential solution, as DP can be applied to a wide range of real-world scenarios where information needs to be sanitized. However, strong indications suggest that this method of quantifying privacy is not always the optimal choice (see e. g. [8]). Furthermore, even if privacy is quantified using various flavors of DP—such as ϵ-DP, (ϵ, δ)-DP, Pufferfish Privacy [36], or Blowfish Privacy [30]—it remains highly debatable how to accurately assess the actual privacy loss that may result from the chosen threshold according to the specific DP definition used (see e. g. [50]). For these reasons, we consider extending our protocol Π_{OutComp} to achieve quantifiable output privacy to be outside the scope of this work. Similar to the selection of authentication and authorization mechanisms, where different use cases (even for the same function f) require different approaches, the choice of an output privacy mechanism for sanitizing results depends heavily on the specific requirements and real-world circumstances of the application implementing the protocol.

We recommend that any implementation of an instantiation using a function f should ensure transparency to input clients about the function f that will be computed on their inputs. This won't solve the problem, but it may bolster a discussion with those directly affected, who may have a better understanding of how the output of f interferes with their privacy.

6 Outsourced Private Set Intersection

In order to demonstrate how the previous discussions from Section 5 apply to the design process of cryptographically securing an application using an instantiation of Π_{OutComp}, we would like to present a case study of our own. The use case for this study is a web service that matches e. g. learners with learning offering providers and themselves. This case study has some notable requirements, which we gathered through interviews with members of the project BIRD[6], an initiative aimed at digitalizing education services in Germany. We will first describe the plain service, without any cryptographic measures beyond secure channels and authentication, and then outline the requirements this service fails to meet from the perspective of threat modeling as identified by key members of the project. Finally, we will explain how a PSI-based protocol can enable the functionality for the matching service, discuss how related work on outsourced PSI falls short of critical requirements, and show how an instantiation of Π_{OutComp} with a strawman PSI protocol successfully meets all security and performance requirements.

[6] `https://www.daad.de/en/the-daad/what-we-do/digitalisation/bird/`

6.1 Use-Case

The basic architecture consists of a centralized server with low-resource user clients, i. e., the learners or students, and low-resource education service provider clients. For brevity, we will limit the use case of education services to student tutoring services. Therefore, the following search criteria is used for matching of a learner with a student tutoring service:

- Role: Teacher or Student
- Federal State: Baden-Württemberg, Bavaria, Berlin and others
- School subjects: German, Mathematics, Geography, English, Physics and others.
- School Type: Primary School, Secondary School and others
- Grade Level: 1 - 13

The basic version of this service can be implemented in a straightforward manner without the need for special cryptographic measures, provided that secure channels, proper authentication, and implementation security are ensured. However, there are certain security requirements that this basic version does not fulfill. This raises concerns that risks, should they materialize, could lead to significant reputational damage for the provider. Therefore, additional measures are necessary to enhance the service, addressing these concerns without at the same time compromising other requirements, such as functionality and performance. Our choice was a Private Set Intersection protocol since this is a well understood primitive and there exists a vast body of literature. We provide a summary of relevant requirements in Appendix D that a PSI protocol should suffice in order to being usable in our case study. However, we found the literature surprisingly unsuitable for meeting all of the requirements simultaneously. This observation motivated an instantiation of Π_{OutComp} with a strawman PSI protocol that we will describe later on.

6.2 Related Work on Outsourced PSI

To the best of our knowledge most of the literature focuses on protocols where only one server is involved and therefore collusion with the only-existing server is explicitly not assumed. Also many of those protocols are highly interactive. Interactiveness means that the input clients need to be online more than once and sometimes at the same time in order to communicate with each other. This may be useful in other applications, such as when input clients must explicitly approve their inputs being used together with inputs from specific other clients. However, in our case, this is not an issue. Also, there are plenty of applications and use cases with systems that are always online, where the non-interactiveness may not be a necessary functional requirement. In such cases, the following works may be more suitable than our work. Such outsourced PSI works are Miyaji et al. [42] and Wang et al. [59]

A special category of such interactive protocols are multi-key homomorphic encryption protocols or threshold homomorphic encryption schemes, which inherently require clients to be online during the decryption stage and perform

a collaborative protocol in addition to the communication in the outsourcing phase, such as in [19,44,9].

The second line of related outsourced PSI constructions assume the non-collusion between the single server and any client, which we deem dangerous in the setting of our case study. This is unsurprising, as PSI is an MPC protocol originally designed for two or more *peer* parties. Having a centralized server with no inputs of its own and no legitimate reason to learn any information about the output of the PSI can lead to the server learning this information, due to the collusion with an input client or an initiating client. One could argue that this, in some way, is detrimental to the original MPC security definition. As a result, scenarios where adversaries can corrupt both a server and a client simultaneously are typically excluded, and protection measures for such cases are usually not described. This leads to many of these protocols relying on the assumption of non-collusion between server and client, which, as discussed, introduces a single point of failure. However, in real-world settings where the threshold for client participation should be as low as possible, administrators of the server infrastructure–who can create accounts themselves and thus access a dummy client's state–could carry out this attack, violating the passive server-client non-collusion assumption. The following works deliberately do not consider this attack, such as those by [55,2,3,4,49,1,33,25,53,35].

The remaining works fall into the category of miscellaneous protocols, which are not suitable for our application or incomparable for various reasons. Many of these works lack a (simulation-based) proof, despite having theorems described, making it difficult to compare them in terms of their adversary model. One such work is by Zheng et al. [62], where they require a dedicated trusted third party, which is outside the scope of our use case. This argument may seem peculiar because we require a key registration with knowledge in Π_{OutComp}, a specialized variant of a public key infrastructure, which is itself a trusted third party. However, we argue that such variants of trusted third parties are more standard and already widely available in the real-world. While Ali et al. [7] provide an extensive proof, their threat/adversary model is highly specific. In their model, the authors distinguish between unauthorized and authorized clients, which is not applicable to our scenario. Furthermore, they assume a trusted third party, referred to as a certificate authority (CA), but this differs significantly from the standard definition. In their case, the CA generates key pairs for the clients, which is not the case in our key registration with knowledge, as the key generation is realized within the real protocol by the parties themselves. Given that our security definition is non-trivial, we exclude works that do not provide a proper proof, as this hinders a meaningful comparison with our own work, i. e., [47,45]

Overall, we have identified one heuristic that seems to be either having a partially interactive outsourced PSI protocol that is secure against server-client collusions, or excluding server-client collusions and, in return, being fully non-interactive after the outsourcing phase. This is unsurprising, as the server-client non-collusion assumption allows secret information to be hidden within the client's state, which can be used to protect all clients from an honest-but-

curious server. Our protocol, on the other hand, achieves full non-interactiveness while also being secure in the case of a server-client collusion, which we consider more dangerous in our use case than server-server collusions. Also, to the best of our knowledge, there does not exist one OPSI protocol that was proven within the UC model but rather in the weaker real-ideal simulation paradigm (a. k. a. standalone security or sequential composability model).

6.3 One-Sided Two-Party Sender-Private PSI

Functionality $\mathcal{F}_{\mathsf{2P-PSI}}$

The functionality is parameterized by the intersection function over two sets $\cap : X \times Y \to Z$ with $|Z| \leq min\{|X|, |Y|\}$. The functionality interacts with two parties, the sender S and the receiver R.

Execution

- When getting input set X from S and input set Y from R, compute the set $Z = X \cap Y$ and output Z to R.

Fig. 5. Two-Party Functionality $\mathcal{F}_{\mathsf{2P-PSI}}$

We employ the well known strawman PSI protocol. The protocol can be seen in Figure 6.

Theorem 3 *Given an IND-CPA^D secure somewhat homomorphic encryption scheme* $\mathsf{HE} = (\mathsf{GEN}, \mathsf{ENC}, \mathsf{DEC})$*, the protocol* $\Pi_{\mathsf{2P-PSI}}$ *securely realizes* $\mathcal{F}_{\mathsf{2P-PSI}}$ *under sequential composition in the presence of a semi-honest adversary* $\mathcal{A}$*.*

For a better understanding we give a short formal proof in Appendix C that shows the simple protocol to securely realize the functionality $\mathcal{F}_{\mathsf{2P-PSI}}$ shown in Figure 5 under sequential composition.

6.4 Outsourced PSI Computation for a Tutoring Service and Community Platform

In the beginning of this section we talked about the different questions that should be addressed in order to instantiate Π_{OutComp} in a secure way. In the following, we will apply all of these considerations to our case study of PSI and the use case of tutoring services.

On the choice of f_1, f_2 The initiating party, $\mathsf{P_{Y,j}}$, is an individual, such as a student, who seeks a suitable tutoring service or another person willing to learn together. These services, or individuals, serve as the input clients $\mathsf{P_{X,i}}$, meaning

Protocol $\Pi_{\mathsf{2P-PSI}}$

Private inputs: The sender party S has input $\mathbf{x}$ and auxiliary information pk for a somewhat homomorphic encryption scheme HE and the receiver party R has input $\mathbf{y}$ and has auxiliary information $(\mathsf{sk}, \mathsf{pk})$. Both input vectors follow a publicly known vector structure for element-wise comparison.
Public inputs: Public parameters of HE
Outputs: R outputs $\mathbf{z} = \mathbf{x} \cap \mathbf{y}$.

Protocol computation:

1. R encrypts its input $c_{\mathbf{y}} = \mathsf{ENC}(\mathsf{pk}, \mathbf{y})$ and sends $(pk, c_{\mathbf{y}})$ to S.
2. When S receives a message $(\mathsf{pk}, c_{\mathbf{y}})$ from R, S encrypts its input vector $c_{\mathbf{x}} = \mathsf{ENC}(\mathsf{pk}, \mathbf{x})$, draws a random vector $c_{\mathbf{r}} \leftarrow \mathcal{C}$ from the HE ciphertext space and computes $c_{\mathbf{S}} = (c_{\mathbf{x}} - c_{\mathbf{y}}) \cdot c_{\mathbf{r}}$; S then sends $c_{\mathbf{S}}$ to R.
3. When R receives the ciphertext $c_{\mathbf{S}}$, it decrypts $c_{\mathbf{S}}$ to $\mathbf{z}' = \mathsf{DEC}(\mathsf{sk}, c_{\mathbf{S}})$; Then it goes elementwise through $\mathbf{z}'$ and checks $\mathbf{z}'_i == 0$ for $i \in [0, \cdots, n-1]$. If this is the case, R adds $\mathbf{y}_i$ to an initially empty initialized set $\mathbf{z}$ and outputs $\mathbf{z}$ at the end.

Fig. 6. the standalone secure 2-party protocol $\Pi_{\mathsf{2P-PSI}}$

we require a computation that accounts for all inputs including the initiator's input. This leads us directly to the outsourced SFE case from **??**, since $f_2 \neq id$. Referring to the $\Pi_{\mathsf{2P-PSI}}$ protocol in Figure 6, we observe in step 3 that the receiver must perform non-trivial comparisons after decryption. This confirms that we are indeed dealing with the proper two-party sender-private SFE case, where a sound proof for standalone simulation-based security is required. The relevant theorem is Theorem 3, and the corresponding proof can be found in Appendix C.

The function f_1 on the other hand is a simple aggregation of all ciphertexts from the input clients $\mathsf{P_{X,i}}$ into one or more ciphertexts, depending on the parameterization and number of clients, in order to compute the result in a SIMD fashion. Thus, we obtain a secure instantiation of Π_{OutComp}. The proof for this instantiation then follows directly from Theorem 2 and Theorem 3.

Output Privacy Our requirement is that the $\mathsf{P_{Y,j}}$ learns the exact intersection of their and the input clients' interests and criteria. Any sanitization applied to the intersection result would reduce its utility to zero in our setting. Therefore, we exclude such methods from this case study. Nevertheless, we do not rule out that there may be other applications of PSI that do not have the same strict requirements, where such methods could be applicable.

Choice of Parties and the Adversary Model There are multiple architecture options that minimize the risk of violating the non-collusion assumption between the providers of $\mathsf{S_C}$ and $\mathsf{S_D}$.

The best option for $\mathsf{S_C}$ in this instantiation is a dedicated (honest-but-curious) third party with sufficient computational capacity to reliably compute f_1 and parts of f_2. Given that the plaintext data constitutes a concise catalog of tutoring services and individuals seeking their services, we do not expect storage requirements or bandwidth to be significant bottlenecks. Therefore, the operating costs of $\mathsf{S_C}$ should primarily be driven by CPU consumption. We propose that such an operator could be a small or medium-sized company, ideally dedicated to providing the sole service of $\mathsf{S_C}$, which, as mentioned earlier, would significantly increase the costs associated with any misbehavior by the entity.

For $\mathsf{S_D}$, there are two options, with a preference for the first. Either the operator of $\mathsf{S_D}$ is the actual operator of the centralized plain version of the education services platform, or the $\mathsf{S_D}$ component is a properly configured TEE within the operator of $\mathsf{S_C}$. In the second case, the operator of $\mathsf{S_C}$ can also be the actual operator of the education services platform. However, we advise against this, as such operators typically lack the necessary expertise to properly configure, administer, and update $\mathsf{S_C}$ in line with the latest advancements in the field. The option of merging $\mathsf{S_D}$ and $\mathsf{P_{Y,j}}$ does not work in this scenario, as there are many different initiators who wish to perform computations over the inputs of the same input clients.

6.5 Evaluation

The following section presents and discusses the results from the evaluation.

IND-CPA^D Attacks **For our implementation, we use the BGV scheme from the Golang library Lattigo[7]. As of January 8, 2025, the IND-CPA$^{\mathrm{D}}$ attacks (see section 5.1) are not addressed in the library's. We decided to keep the evaluation unchanged because the related work discussed in section 6.2 also did not consider these attacks in their evaluations. We will use this eprint version to communicate to the developers the need of addressing these attacks. Once the library releases newer versions of the BGV implementation with effective countermeasures in place, we will update this evaluation with metrics based on an IND-CPA$^{\mathrm{D}}$ secure implementation of BGV.**

Setup and Parameterization All evaluations were performed on a server with an AMD EPYC 7763 64-Core Processor, albeit only 1 core was effectively utilized due to missing parallelization in the benchmark. The benchmark implementation is available on GitHub.[8]. Since our protocol involves only one multiplication, we selected a simple parameter set with a ternary distribution, specifically $log_2(N) = 13$ and $log_2(Q) = 58$, based on recommendations from [5], achieving a 256-bit security level. Furthermore, our plaintext space, which consists of entries from matching criteria such as ZIP codes, is sufficiently small to fit within a

[7] `https://github.com/tuneinsight/lattigo`
[8] `https://github.com/collapsinghierarchy/opsi_sepdutyhe_bench`

26-bit plaintext modulus. For accommodating larger plaintext spaces that cover e. g. 256-bit hash values one will have to use a different parameterization or employ a Chinese Remainder Theorem in order to encode the hash value into smaller components.

Table 1. Performance metrics for the Calculator, Decryptor, and Initiator averaged over 100 repetitions. 'Slots' indicates that these measurements apply to any combination of the number of clients and the number of criteria per client. For example, 524288 slots can represent the performance of matching 32768 clients, each with 16 criteria, simultaneously.

Slots	**Calculator (w/Aggr.)**	**Calculator (w/oAggr.)**	**Decryptor**	**Initiator**
2048	14.71ms	1.02ms	326µs	14µs
4096	26.45ms	1.02ms	323µs	14µs
8192	49.42ms	1.00ms	317µs	13µs
16384	104.76ms	1.99ms	636µs	26µs
32768	248.24ms	3.92ms	1.31ms	68µs
65536	382.02ms	7.76ms	3.25ms	133µs
131072	1.01s	16.22ms	5.22ms	299µs
262144	5.58s	35.17ms	11.87ms	453µs
524288	14.22s	66.59ms	22.34ms	928µs

Table 2. Storage metrics for the Calculator and Initiator given the parameterization of $log_2(N) = 13$ and $log_2(Q) = 58$.

Slots	**Storage (Calc.)**	**Storage (Init. mask)**
32768	237.58 kB	106.5 kB
65536	475.14 kB	213 kB
131072	950.28 kB	426 kB
262144	1.9 MB	852 kB
524288	3.8 MB	1.7 MB

Evaluation Results and Discussion The results are summarized in Table 1. Overall, the results show that the CPU consumption within the calculator grows linearly with the number of slots. We have excluded the evaluation of the input client's performance, as the overhead involves only a single encryption. The initiating party, on the other hand, requires slightly more resources than a single encryption, as it also must sample and encrypt a large mask that matches the size of the final result. Since the final result includes the complete (randomized) services catalog and the learning partner register, the mask typically spans multiple ciphertexts. Readers may wonder why we employ homomorphic encryption instead of simply transferring the complete catalog in plaintext, which would

effectively result in a trivial Private Information Retrieval protocol without any encryption. The reason is that some matching partners may be private individuals who do not wish their data to be shared with anyone outside of the matched parties. Future improvements, however, may focus on reducing the size of the mask on the initiating party's side. One such possible improvement is employing a PRNG to generate the mask from a small seed such that the initiator only has to store the small seed. The interviews have indicated that the BIRD[9] project consortium anticipates the number of input clients to range between 10^5 and 10^6 participants. The evaluations suggest that performance is more than sufficient for these numbers. Furthermore, the aggregation step significantly reduces storage consumption, leaving bandwidth as the only potential bottleneck. However, we do not expect the outsourcing phase to occur all at once and thus, we do not anticipate this bottleneck to pose a significant issue.

7 Conclusion and Future Work

In this work, we proposed a general construction for outsourcing secure function evaluations without having to assume that no client can collude with one server and without requiring interactions among the clients. Our protocol follows the separation of duties principle: Since we assume that the two servers do not collude, one server, the calculator server, is used for storing the encrypted outsourced clients' inputs and calculating the encrypted interim results whereas the other server, the decryptor, is used for decrypting the interim results. We employ a mask, only known to the initiating party, in order to prevent the decryptor learning the results. This results in the clients only having to encrypt their inputs for participating in the computation. We furthermore were able to prove the security of our construction within the UC framework by only requiring a sequentially composable (i. e. standalone secure) one-sided two-party secure function evaluation protocol which is based on a somewhat homomorphic encryption scheme. We also showed the usability of our construction by giving an instantiation example of a tutoring services platform using Private Set Intersection accompanied by an implementation in go and an evaluation of the implementation's efficiency.

Since we concentrated on a practical solution secure against passive corruptions, a next step could be in finding a practical solution that is secure against active adversaries. The simplest solution would be adding zero-knowledge proofs to each shared message but to the best of our knowledge, zero knowledge proofs are not yet practically usable in real-world applications. Currently, the function to be evaluated on is already fixed in the beginning. Therefore it could be another interesting research direction to find out how to adjust different user-chosen functions for every computation round such that an ad-hoc computation over these functions could be realized, similar to the approach by López-Alt et al. [40].

[9] `https://www.daad.de/en/the-daad/what-we-do/digitalisation/bird/`

Acknoledgements

This work was partially funded by the European Union -NextGenerationEU through the German Federal Ministry of Education and Research (BMBF) as part of the Digital Education initiative under grant number 16NB001. The views and opinions expressed are solely those of the authors and do not necessarily reflect the views of the European Union, European Commission or the Federal Ministry of Education and Research. Neither the European Union, the European Commission nor the Federal Ministry of Education and Research can be held responsible for them.

This work was also partially funded by the German Federal Ministry of Education and Research (BMBF) under the projects *ANYMOS* and *PRIVILEG*.

We would like to thank to anonymous reviewers of PKC2025 for constructive feedback and pointing out that IND-CPA^D has to be incorporated into our framework as well.

Finally, we would like to thank Prof. Ulrike Lucke from the university of Potsdam and the project BIRD[10] for valueable insights into their requirements and giving us space to develop this solution.

References

1. Abadi, A., Dong, C., Murdoch, S.J., Terzis, S.: Multi-party updatable delegated private set intersection. In: Eyal, I., Garay, J.A. (eds.) FC 2022. LNCS, vol. 13411, pp. 100–119. Springer, Cham (May 2022). https://doi.org/10.1007/978-3-031-18283-9_6
2. Abadi, A., Terzis, S., Dong, C.: O-PSI: Delegated private set intersection on outsourced datasets. In: Federrath, H., Gollmann, D. (eds.) ICT Systems Security and Privacy Protection. IFIPAICT, vol. 455, pp. 3–17. Springer, Cham (May 2015). https://doi.org/10.1007/978-3-319-18467-8_1
3. Abadi, A., Terzis, S., Dong, C.: VD-PSI: Verifiable delegated private set intersection on outsourced private datasets. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 149–168. Springer, Berlin, Heidelberg (Feb 2017). https://doi.org/10.1007/978-3-662-54970-4_9
4. Abadi, A., Terzis, S., Metere, R., Dong, C.: Efficient delegated private set intersection on outsourced private datasets. IEEE Transactions on Dependable and Secure Computing **16**(4), 608–624 (may 2017). https://doi.org/10.1109/TDSC.2017.2708710
5. Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., Lokam, S., Micciancio, D., Moody, D., Morrison, T., Sahai, A., Vaikuntanathan, V.: Homomorphic encryption security standard. Tech. rep., HomomorphicEncryption.org (Nov 2018)
6. Alexandru, A., Al Badawi, A., Micciancio, D., Polyakov, Y.: Application-aware approximate homomorphic encryption: Configuring fhe for practical use. Cryptology ePrint Archive (2024)

[10] https://www.uni-potsdam.de/de/multimedia/projekte/bildungsraum-digital-bird

7. Ali, M., Mohajeri, J., Sadeghi, M.R., Liu, X.: Attribute-based fine-grained access control for outscored private set intersection computation. J: InS **536**, 222–243 (Oct 2020). https://doi.org/10.1016/j.ins.2020.05.041
8. Bambauer, J., Muralidhar, K., Sarathy, R.: Fool's gold: An illustrated critique of differential privacy. Vanderbilt Journal of Entertainment & Technology Law **16**(4), 701–755 (Oct 2014)
9. Bay, A., Erkin, Z., Hoepman, J.H., Samardjiska, S., Vos, J.: Practical multi-party private set intersection protocols. TIFS 2021 **17**, 1–15 (Oct 2021). https://doi.org/10.1109/TIFS.2021.3118879
10. Beskorovajnov, W., Dörre, F., Hartung, G., Koch, A., Müller-Quade, J., Strufe, T.: ConTra corona: Contact tracing against the coronavirus by bridging the centralized-decentralized divide for stronger privacy. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part II. LNCS, vol. 13091, pp. 665–695. Springer, Heidelberg (Dec 2021). https://doi.org/10.1007/978-3-030-92075-3_23
11. Beskorovajnov, W., Gröll, R., Müller-Quade, J., Ottenhues, A., Schwerdt, R.: A new security notion for PKC in the standard model: Weaker, simpler, and still realizing secure channels. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) PKC 2022, Part II. LNCS, vol. 13178, pp. 316–344. Springer, Cham (Mar 2022). https://doi.org/10.1007/978-3-030-97131-1_11
12. Bogetoft, P., Christensen, D.L., Damgård, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J.D., Nielsen, J.B., Nielsen, K., Pagter, J., Schwartzbach, M.I., Toft, T.: Secure multiparty computation goes live. In: Dingledine, R., Golle, P. (eds.) FC 2009. LNCS, vol. 5628, pp. 325–343. Springer, Berlin, Heidelberg (Jan 2009). https://doi.org/10.1007/978-3-642-03549-4_20
13. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Annual cryptology conference. pp. 868–886. Springer (2012)
14. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
15. Bresson, E., Catalano, D., Pointcheval, D.: A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In: Laih, C.S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 37–54. Springer, Berlin, Heidelberg (Nov / Dec 2003). https://doi.org/10.1007/978-3-540-40061-5_3
16. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145 (Oct 2001). https://doi.org/10.1109/SFCS.2001.959888
17. Canetti, R.: Universally composable security. J. ACM **67**(5), 94 (Sep 2020). https://doi.org/10.1145/3402457
18. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: CCS 2017. pp. 1243–1255. CCS '17', Association for Computing Machinery (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134061
19. Chen, L., Li, Z., Chen, Z., Liu, Y.: Two anti-quantum attack protocols for secure multiparty computation. In: Zhang, H., Zhao, B., Yan, F. (eds.) CTCIS 2018. CCIS, vol. 960, pp. 338–359. Springer, Singapore (Jan 2019). https://doi.org/10.1007/978-981-13-5913-2_21
20. Cheon, J.H., Choe, H., Passelègue, A., Stehlé, D., Suvanto, E.: Attacks against the ind-cpad security of exact fhe schemes. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. pp. 2505–2519 (2024)

21. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part I. LNCS, vol. 10624, pp. 409–437. Springer, Cham (Dec 2017). https://doi.org/10.1007/978-3-319-70694-8_15
22. Chillotti, I., Gama, N., Georgieva, M., Izabachene, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22. pp. 3–33. Springer (2016)
23. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. Journal of the ACM (JACM) **45**(6), 965–981 (Nov 1998). https://doi.org/10.1145/293347.293350
24. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Berlin, Heidelberg (Aug 2012). https://doi.org/10.1007/978-3-642-32009-5_38
25. Debnath, S.K., Sakurai, K., Dey, K., Kundu, N.: Secure outsourced private set intersection with linear complexity. In: DSC 2021. pp. 1–8. IEEE (Feb 2021). https://doi.org/10.1109/DSC49826.2021.9346230
26. Dwork, C.: Differential privacy. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) Automata, Languages, and Programming. LNCS, vol. 4052, pp. 1–12. Springer, Berlin, Heidelberg (Jul 2006). https://doi.org/10.1007/11787006_1
27. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive (2012)
28. Garimella, G., Rosulek, M., Singh, J.: Structure-aware private set intersection, with applications to fuzzy matching. In: Annual International Cryptology Conference. pp. 323–352. Springer (2022)
29. Goldwasser, S., Micali, S.: Probabilistic encryption and how to play mental poker keeping secret all partial information. In: 14th ACM STOC. pp. 365–377. ACM Press (May 1982). https://doi.org/10.1145/800070.802212
30. He, X., Machanavajjhala, A., Ding, B.: Blowfish privacy: Tuning privacy-utility trade-offs using policies. In: SIGMOD 2014. pp. 1447–1458. SIGMOD '14, Association for Computing Machinery (Jun 2014). https://doi.org/10.1145/2588555.2588581
31. Jain, A.K.: Corruption: A review. Journal of Economic Surveys **15**(1), 71–121 (Feb 2001). https://doi.org/10.1111/1467-6419.00133
32. Jakobsen, T.P., Nielsen, J.B., Orlandi, C.: A framework for outsourcing of secure computation. In: CCSW 2014. p. 81–92. CCSW '14, Association for Computing Machinery (Nov 2014). https://doi.org/10.1145/2664168.2664170
33. Jolfaei, A.A., Mala, H., Zarezadeh, M.: EO-PSI-CA: Efficient outsourced private set intersection cardinality. J. JISA **65**(102996), 11 (Mar 2022). https://doi.org/https://doi.org/10.1016/j.jisa.2021.102996
34. Kamara, S., Mohassel, P., Raykova, M.: Outsourcing multi-party computation. Cryptology ePrint Archive, Report 2011/272 (2011), https://eprint.iacr.org/2011/272
35. Kamara, S., Mohassel, P., Raykova, M., Sadeghian, S.: Scaling private set intersection to billion-element sets. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 195–215. Springer, Berlin, Heidelberg (Mar 2014). https://doi.org/10.1007/978-3-662-45472-5_13
36. Kifer, D., Machanavajjhala, A.: Pufferfish: A framework for mathematical privacy definitions. TODS **39**(1), 1–36 (Jan 2014). https://doi.org/10.1145/2514689

37. Knott, B., Venkataraman, S., Hannun, A., Sengupta, S., Ibrahim, M., van der Maaten, L.: Crypten: Secure multi-party computation meets machine learning. In: Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems. vol. 34, pp. 4961–4973. Curran Associates, Inc. (Dec 2021)
38. Li, B., Micciancio, D.: On the security of homomorphic encryption on approximate numbers. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 648–677. Springer (2021)
39. Lopez-Alt, A., Tromer, E., Vaikuntanathan, V.: Cloud-assisted multiparty computation from fully homomorphic encryption. Cryptology ePrint Archive, Report 2011/663 (2011), https://eprint.iacr.org/2011/663
40. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: STOC 2012. p. 1219–1234. STOC '12, Association for Computing Machinery (May 2012). https://doi.org/10.1145/2213977.2214086
41. McSherry, F., Talwar, K.: Mechanism design via differential privacy. In: 48th FOCS. pp. 94–103. IEEE (Oct 2007). https://doi.org/10.1109/FOCS.2007.66
42. Miyaji, A., Nakasho, K., Nishida, S.: Privacy-preserving integration of medical data. Journal of Medical Systems **41**(37), 10 (Jan 2017). https://doi.org/10.1007/s10916-016-0657-4
43. Mouchet, C., Bertrand, E., Hubaux, J.P.: An efficient threshold access-structure for RLWE-based multiparty homomorphic encryption. Journal of Cryptology **36**(2), 10 (Mar 2023). https://doi.org/10.1007/s00145-023-09452-8
44. Mouchet, C., Troncoso-Pastoriza, J., Bossuat, J.P., Hubaux, J.P.: Multiparty homomorphic encryption from ring-learning-with-errors. PoPETs **2021**(4), 291–311 (Jun 2021). https://doi.org/10.2478/popets-2021-0071
45. Oliaiy, M.M., Ameri, M.H., Mohajeri, J., Aref, M.R.: A verifiable delegated set intersection without pairing. In: IranianCEE 2017. pp. 2047–2051. IEEE (May 2017). https://doi.org/10.1109/IranianCEE.2017.7985395
46. Peter, A., Tews, E., Katzenbeisser, S.: Efficiently outsourcing multiparty computation under multiple keys. IEEE Transactions on Information Forensics and Security **8**(12), 2046–2058 (Dec 2013). https://doi.org/10.1109/TIFS.2013.2288131
47. Qiu, S., Dai, Z., Zha, D., Zhang, Z., Liu, Y.: PPSI: Practical private set intersection over large-scale datasets. In: SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI 2019. pp. 1249–1254. IEEE (Aug 2019). https://doi.org/10.1109/SmartWorld-UIC-ATC-SCALCOM-IOP-SCI.2019.00232
48. Rivest, R.L., Adleman, L., Dertouzos, M.L.: On data banks and privacy homomorphisms. In: Lipton, R., Dobkin, D., Jones, A. (eds.) Foundations of secure computation, pp. 171–189. Academic Press, Inc. (1978)
49. Ruan, O., Huang, X., Mao, H.: An efficient private set intersection protocol for the cloud computing environments. In: BigDataSecurity, HPSC and IDS 2020. pp. 254–259. IEEE (May 2020). https://doi.org/10.1109/BigDataSecurity-HPSC-IDS49724.2020.00053
50. Rupp, V., von Grafenstein, M.: Clarifying "personal data" and the role of anonymisation in data protection law including and excluding data from the scope of the gdpr (more clearly) through refining the concept of data protection. J. CLSR **52**(105932), 25 (Apr 2024). https://doi.org/10.1016/j.clsr.2023.105932
51. Schoenmakers, B., Veeningen, M., de Vreede, N.: Trinocchio: Privacy-preserving outsourcing by distributed verifiable computation. In: Manulis, M., Sadeghi, A.R., Schneider, S. (eds.) ACNS 2016. LNCS, vol. 9696, pp. 346–366. Springer, Cham (Jun 2016). https://doi.org/10.1007/978-3-319-39555-5_19

52. Stefanov, E., Shi, E.: Multi-cloud oblivious storage. In: CCS 2013. pp. 247–258. CCS '13, Association for Computing Machinery (Nov 2013). https://doi.org/10.1145/2508859.2516673
53. Tajima, A., Sato, H., Yamana, H.: Outsourced private set intersection cardinality with fully homomorphic encryption. In: ICMCS 2018. pp. 1–8. IEEE (May 2018). https://doi.org/10.1109/ICMCS.2018.8525881
54. Talviste, R.: Applying secure multi-party computation in practice. Ph. D. dissertation (2016), https://core.ac.uk/works/46776929/
55. Terada, S., Yoneyama, K.: Improved verifiable delegated private set intersection. In: ISITA 2018. pp. 520–524. IEEE (Oct 2018). https://doi.org/10.23919/ISITA.2018.8664310
56. Thangam, V., Chandrasekaran, K.: Elliptic curve based secure outsourced computation in multi-party cloud environment. In: Mueller, P., Thampi, S.M., Bhuiyan, M.Z.A., Ko, R., Doss, R., Calero, J.M.A. (eds.) Security in Computing and Communications. CCIS, vol. 625, pp. 199–212. Springer, Singapore (Sep 2016). https://doi.org/10.1007/978-981-10-2738-3_17
57. Veugen, T., de Haan, R., Cramer, R., Muller, F.: A framework for secure computations with two non-colluding servers and multiple clients, applied to recommendations. IEEE Transactions on Information Forensics and Security **10**(3), 445–457 (Nov 2015). https://doi.org/10.1109/TIFS.2014.2370255
58. Wang, B., Li, M., Chow, S.S.M., Li, H.: Computing encrypted cloud data efficiently under multiple keys. In: CNS 2013. pp. 504–513. IEEE (Oct 2013). https://doi.org/10.1109/CNS.2013.6682768
59. Wang, X.A., Xhafa, F., Luo, X., Zhang, S., Ding, Y.: A privacy-preserving fuzzy interest matching protocol for friends finding in social networks. Soft Computing **22**, 2517–2526 (Feb 2018). https://doi.org/10.1007/s00500-017-2506-x
60. Wolf, M.J., Miller, K., Grodzinsky, F.S.: Why we should have seen that coming: comments on microsoft's tay experiment, and wider implications. ACM SIGCAS Computers and Society **47**(3), 54–64 (Sep 2017). https://doi.org/10.1145/3144592.3144598
61. Yasuda, M.: Secure hamming distance computation for biometrics using ideal-lattice and ring-lwe homomorphic encryption. Information Security Journal: A Global Perspective **26**(2), 85–103 (2017)
62. Zheng, Q., Xu, S.: Verifiable delegated set intersection operations on outsourced encrypted data. In: IC2E 2015. pp. 175–184. IEEE (Mar 2015). https://doi.org/10.1109/IC2E.2015.38

A Security

Let $\mathcal{A}$ be a semi-honest, static adversary that interacts with parties performing the protocol Π_{OutComp} shown in Figure 2. We construct an ideal adversary $\mathcal{S}$ for the ideal protocol interacting with $\mathcal{F}_{\mathsf{OutComp}}$ such that no environment $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and the protocol Π_{OutComp} or with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{OutComp}}$.

Simulating the communication with $\mathcal{Z}$: Every input value that $\mathcal{S}$ receives from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape (as if coming from $\mathcal{A}$'s environment). Every output value written by $\mathcal{A}$ on its output tape is copied to $\mathcal{S}$'s own output tape (to be read by $\mathcal{S}$'s environment $\mathcal{Z}$).

Throughout the whole proof, we are assuming the dummy-adversary which simply forwards in the real-world execution all corruption commands coming from $\mathcal{Z}$ to the corrupted parties and all messages coming from the corrupted parties back to $\mathcal{Z}$. In the ideal-world execution, $\mathcal{S}$ receives all corruption messages from $\mathcal{Z}$ and sends all simulated protocol messages to $\mathcal{Z}$. All inputs to the simulated functionality $\mathcal{F}_{\mathsf{KRK}}$ sent by $\mathcal{Z}$ are directly provided to $\mathcal{S}$. Additionally, $\mathcal{S}$ provides all (modified) inputs of the corrupted dummy parties (sent by $\mathcal{Z}$) to the ideal functionality $\mathcal{F}_{\mathsf{OutComp}}$ and $\mathcal{Z}$ receives the outputs from $\mathcal{F}_{\mathsf{OutComp}}$ through the dummy parties.

A.1 Corrupted Initiator

In the case of a corrupted initiator $\mathsf{P}_{\mathsf{Y,j}}$, the constructed simulator $\mathcal{S}$ works as follows:

1. Setup Phase: When getting the message $(\mathsf{retrieve}, \mathsf{sid}, \mathsf{S}_{\mathsf{D}})$ from $\mathcal{Z}$ to $\mathcal{F}_{\mathsf{KRK}}$, generate a fresh key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{GEN}(1^{\lambda})$ and simulate receiving the message $(\mathsf{retrieve}, \mathsf{sid}, \mathsf{pk}, \mathsf{S}_{\mathsf{D}})$ from $\mathcal{F}_{\mathsf{KRK}}$.
2. Outsourcing phase: When receiving a message $(\mathsf{outsource}, c_{y,j}, c_{r,j})$ from $\mathsf{P}_{\mathsf{Y,j}}$, extract the input y_j and mask r_j by decrypting the ciphertexts $y_j = \mathsf{DEC}(\mathsf{sk}, y_j)$ and $r_j = \mathsf{DEC}(\mathsf{sk}, r_j)$. Then send y_j to $\mathcal{F}_{\mathsf{OutComp}}$, store r_j and simulate forwarding the received message to S_{C}.
3. Computation phase: Whenever getting the message $(\mathsf{start}, \mathsf{sid}, \mathsf{ssid})$ from $\mathcal{Z}$, it forwards this message to $\mathcal{F}_{\mathsf{OutComp}}$.
4. Result retrieval: When getting the message $(\mathsf{output?}, \mathsf{sid}, \mathsf{ssid})$ from $\mathcal{Z}$, forward this message to $\mathcal{F}_{\mathsf{OutComp}}$. Upon receiving back the result z from $\mathcal{F}_{\mathsf{OutComp}}$, follow the instruction of Π_{2PC}'s simulation to compute the message d, then mask using the extracted r and and simulate receiving $d' = d + r$ from S_{D} to $\mathcal{A}$.

The simulatability of the protocol is reduced to the simulatability of the underlying standalone secure protocol Π_{2PC}, since the simulator works exactly the same when the receiver in Π_{2PC} is corrupted, except masking the input/output of Π_{2PC}. Therefore assume for the sake of contradiction that there exists an environment $\mathcal{Z}$ that is able to distinguish the real-world protocol execution of Π_{OutComp} from the ideal-world simulation. Then we are able to create a non-interactive distinguisher $\mathcal{D}$ that is a able to distinguish a real-world execution of Π_{2PC} and ideal-world simulation with non-negligible probability by using $\mathcal{Z}$ in order to do the distinguishing work. The constructed $\mathcal{D}$ works as follows:

- Emulate a real-world execution of Π_{OutComp} with $\mathcal{Z}$.
- When starting the first computation round, gather all inputs from all input clients in order to compute $x = f_1(x_1, \cdots, x_n)$, the initiator's input y_j and the key pair $(\mathsf{sk}, \mathsf{pk})$ for the decryptor and feed the challenge protocol Π_{2PC} with the given inputs and the key pair as R's auxiliary information and receive the generated transcript (c_r, c_s).

- When $\mathsf{P}_{\mathsf{Y},\mathsf{j}}$ has to get a masked interim result from S_D, take the last encrypted message c_S from the challenge transcript, compute $d' = \mathsf{DEC}(\mathsf{sk}, c_S) + r$ (where r is the masking value generated by the initiator in the outsourcing phase) and simulate reveiving d' from S_D
- The other computation rounds are done as in the real-word execution.
- Output whatever $\mathcal{Z}$ outputs.

To evaluate $\mathcal{D}$'s advantage in distinguishing a simulated execution of Π_{2PC} from a real-world execution, let us look at the case where the transcript $\mathcal{D}$ received is the transcript of a real-world execution. In this case $\mathcal{D}$'s output is identically distributed as $\mathcal{Z}$'s output in the real-world execution.

Now let us look at the case where Π_{2PC}'s transcript is a simulated transcript. In this case, the simulated message $d' = \mathsf{DEC}(\mathsf{sk}, c_S) + r$ forwarded to $\mathcal{Z}$ is done accordingly to the simulation of the outsourced overall protocol and therefore the interaction in this case is the same as in the simulation of the overall outsourced protocol.Therefore $\mathcal{Z}$'s output is identically distributed as in the ideal-world simulation.

Following both cases, $\mathcal{D}$ outputs *ideal* with the same probability as $\mathcal{Z}$ does. Since we assume that $\mathcal{Z}$ is able to distinguish the real-world execution from the ideal-world simulation, it follows that $\mathcal{D}$ is able to distinguish a real protocol transcript from a simulated one with the same probability as $\mathcal{Z}$ is able to distinguish the real-world execution from the ideal world simulation. Since we assume $\mathcal{Z}$ is able to do this with non-negligible probability, this contradicts the assumption that the real-world execution of Π_{2PC} is computationally indistinguishable from its simulation.

A.2 Corrupted Subset of Input Clients

In the case of corrupted subset of input clients, the constructed $\mathcal{S}$ works as follows:

- In the registration phase, whenever receiving a request $(\mathsf{retrieve}, \mathsf{sid}, \mathsf{S}_\mathsf{D})$ from $\mathcal{Z}$ to be delivered to $\mathcal{F}_{\mathsf{KRK}}$, generate a fresh key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GEN}(1^\lambda)$ and simulate receiving it from $\mathcal{F}_{\mathsf{KRK}}$.
- Whenever receiving an encrypted outsourcing message $c_{x,i}$ from $\mathcal{Z}$, extract the corrupted input client's input x_i.

Since the messages in the simulation are simply forwarded, the simulated messages do not differ from the actual messages sent in the real-world protocol execution. Therefore the simulation is perfect.

A.3 Corrupted Initiator and Subset of Input Clients

In the case of corrupted initiator and subset of input clients, the simulator works as follows:

- In the client registration phase, when getting a request $(\mathsf{retrieve}, \mathsf{sid}, \mathsf{S}_\mathsf{D})$ from $\mathcal{Z}$ to $\mathcal{F}_{\mathsf{KRK}}$, generate a key pair and simulate receiving it from $\mathcal{F}_{\mathsf{KRK}}$

- In the outsourcing phase of the input clients, extract x_i the received encrypted input $c_{x,i}$, send x_i to $\mathcal{F}_{\mathsf{OutComp}}$ and simulate sending the encrypted input to the calculator.
- When getting some encrypted input $c_{y,j}$ and the encrypted mask $c_{r,j}$ from $\mathcal{Z}$ in the outsourcing phase of an initiator, decrypt the ciphertexts and send the extracted inputs to the functionality and simulate sending the encrypted input messages to the calculator.
- Output phase: Whenever getting the message $(\mathsf{output?}, \mathsf{sid}, \mathsf{ssid})$ from $\mathcal{Z}$, forward the request to $\mathcal{F}_{\mathsf{OutComp}}$ in order to retrieve the result z. Then follow the last step of the simulation of the underlying Π_{2PC} using z in order to get a plaintext message d. Then compute $d' = d' + r$ and simulate receiving d' from $\mathsf{S_D}$.

The only difference between the messages the simulator generates and the messages that are shared among the real protocol execution is the messages that are generated in simulation step 3: In this step, $\mathcal{S}$ follows the last simulation step of the underlying two-party protocol and masks the decrypted simulated message whereas in the real-world execution the initiator gets the randomized and masked computation result from the decryptor. Here we have to reduce the simulatablity of the whole protocol execution to the underlying standalone secure protocol Π_{2PC}. Therefore assume for the sake of contradiction that there exists an environment $\mathcal{Z}$ that is able to distinguish the real-world execution from the ideal-world simulation. Then we are able to create a non-interactive distinguisher $\mathcal{D}$ that is a able to distinguish a real-world execution of Π_{2PC} and ideal-world simulation with non-negligible probability by using $\mathcal{Z}$ in order to do the distinguishing work. The constructed $\mathcal{D}$ works as follows:

- Emulate a real-world execution of Π_{OutComp} with $\mathcal{Z}$.
- When starting the first computation round, gather all inputs from all input clients in order to compute $x = f_1(x_1, \cdots, x_n)$, the initiator's input y_j and the key pair $(\mathsf{sk}, \mathsf{pk})$ for the decryptor and feed the challenge protocol Π_{2PC} with the given inputs and the key pair as R's auxiliary information and receive the generated transcript c_d.
- When $\mathsf{P_{Y,j}}$ has to get a masked interim result from $\mathsf{S_D}$, take the last encrypted message c_S from the challenge transcript, decrypt it to $d = \mathsf{DEC}(\mathsf{sk}, c_S)$ and compute $d' = d + r$ (where r is the masking value generated by the initiator in the outsourcing phase). Then, simulate receiving d' from $\mathsf{S_D}$.
- The other computation rounds are done as in the real-word execution.
- Output whatever $\mathcal{Z}$ outputs.

To evaluate $\mathcal{D}$'s advantage in distinguishing a simulated execution of Π_{2PC} from a real-world execution, let us look at the case where the transcript $\mathcal{D}$ received is the transcript of a real-world execution. In this case $\mathcal{D}$'s output is identically distributed as $\mathcal{Z}$'s output in the real-world execution.

Now let us look at the case where Π_{2PC}'s transcript is a simulated transcript. In this case the message $d' = \mathsf{DEC}(\mathsf{sk}, c_S) + r$ is generated as in the simulation

and therefore the the interaction as the same as in the simulation. Therefore $\mathcal{Z}$'s output is identically distributed as in the ideal-world simulation.

Following the both cases, $\mathcal{D}$ outputs *ideal* with the same probability as $\mathcal{Z}$ does. Since we assume that $\mathcal{Z}$ is able to distinguish the real-world execution from the ideal-world simulation, it follows that $\mathcal{D}$ is able to distinguish a real protocol transcript from a simulated one with non-negligible probability, which contradicts the assumption that the real-world execution of Π_{2PC} is computationally indistinguishable from its simulation.

A.4 Corrupted Calculator (And Corrupted Subset of Initiators and Input Clients)

In the case of a corrupted calculator, we do not have to differentiate the cases where a subset of initiators or input clients is corrupted since we have to simulate the subset of the honest clients either way. Therefore we present a single simulation: The constructed simulator works as follows:

- In the server initialization phase, when getting the request $(\mathsf{retrieve}, \mathsf{sid}, \mathsf{S_D})$ from $\mathcal{Z}$ to $\mathcal{F}_{\mathsf{KRK}}$, generate a fresh key pair, store it and simulate receiving it from $\mathcal{F}_{\mathsf{KRK}}$.
- In the registration phase, when getting a key request $(\mathsf{retrieve}, \mathsf{sid}, \mathsf{S_D})$ from $\mathcal{Z}$ to $\mathcal{F}_{\mathsf{KRK}}$, take the stored pk and simulate receiving it from $\mathcal{F}_{\mathsf{KRK}}$.
- Whenever getting the encrypted input $c_{y,j}$ and $c_{r,j}$ from $\mathcal{Z}$ for a corrupted $\mathsf{P_{Y,j}}$, extract the input y_j and mask r from the ciphertexts, send y_j to $\mathcal{F}_{\mathsf{OutComp}}$ and store $(\mathsf{P_{Y,j}}, r)$.
- Whenever getting a message $(\mathsf{input}, \mathsf{sid}, \mathsf{ssid}, \mathsf{P_{Y,j}})$ from $\mathcal{F}_{\mathsf{OutComp}}$ for an honest $\mathsf{P_{Y,j}}$, draw two random ciphertexts y_j and r_j from the ciphertext space and simulate receiving it from the honest $\mathsf{P_{Y,j}}$.
- Whenever getting the message $(\mathsf{sid}, \mathsf{pid}, c_{x,i})$ from $\mathcal{Z}$ for a corrupted $\mathsf{P_{X,i}}$ to $\mathsf{S_C}$, decrypt $c_{x,i}$ in order to extract $x_i = \mathsf{DEC}(\mathsf{sk}, c_{x,i})$, forward x_i to $\mathcal{F}_{\mathsf{OutComp}}$ and simulate forwarding the received message to $\mathsf{S_C}$.
- Whenever getting a message $(\mathsf{input}, \mathsf{P_{X,i}})$ from $\mathcal{F}_{\mathsf{OutComp}}$, where $\mathsf{P_{X,i}}$ is an honest input client, draw a random ciphertext $c_{x,i}$ from the ciphertext space and simulate receiving it from $\mathsf{P_{X,i}}$.
- Whenever getting the message $(\mathsf{start}, \mathsf{sid}, \mathsf{ssid}, \mathsf{pid})$ from $\mathcal{Z}$ for a corrupted $\mathsf{P_{Y,j}}$, forward the message to $\mathcal{F}_{\mathsf{OutComp}}$, follow the calculator's protocol instruction in order to compute $c_{d'}$ and simulate sending $c_{d'}$ to $\mathsf{S_D}$.
- Whenever getting the message $(\mathsf{start}, \mathsf{ssid}, \mathsf{P_{Y,j}})$ from $\mathcal{F}_{\mathsf{OutComp}}$ for an honest $\mathsf{P_{Y,j}}$, follow the calculator's protocol instruction in order to compute $c_{d'}$ and simulate sending $c_{d'}$ to $\mathsf{S_D}$.
- Result retrieval: Whenever getting the message $(\mathsf{output?}, \mathsf{ssid})$ from $\mathcal{Z}$ for a corrupted $\mathsf{P_{Y,j}}$, forward the request to $\mathcal{F}_{\mathsf{OutComp}}$ in order to get the result z. Then follow the last simulation step of Π_{2PC} using z to get the last (decrypted) message d, compute $d' = d + r_j$ and simulate receiving d' from $\mathsf{S_D}$.

There are two general differences between the simulation and the real-word execution of the protocol:

1. When an honest client (which is either an input client or an initiator) is told to outsource its input, in the real-word execution the message sent by the clients consists of the encrypted inputs (plus the encrypted masking value of the initiator), whereas in the ideal-world simulation message computed by the simulator consists of random ciphertexts.
2. When a corrupted initiator requests the decryptor for the result, it gets a message computed accordingly to the real-world protocol whereas in the simulation, the message generated by the simulator is computed accordingly to last simulation step of the two-party protocol.

Therefore, we have to differentiate between three hybrid games:

- H_0: real-world execution
- H_1: H_0 with the exception that the ciphertexts $c_{x,i}, c_{y.j}$ and $c_{r,j}$ sent from all honest clients to $\mathsf{S_C}$ are encryptions of random values.
- H_2: ideal-world simulation

To prove the indistinguishability between the hybrid games H_0 and H_1, let us assume for the sake of contradiction that there exists an environment $\mathcal{Z}'$ that is able to distinguish the simulation from the real-world execution. Then we are able to construct an adversary B that is able to break the IND-CPA$^{\mathrm{D}}$ security of the underlying homomorphic encryption. B emulates the real-word execution with $\mathcal{Z}'$. The modified emulation works as follows:

- Simulate the real-word execution, with $\mathcal{Z}'$.
- Initialize an empty set $I = \{\}$ for the indexes of executed oracle requests and set $i := 0$.
- When $\mathcal{Z}'$ requests $\mathcal{F}_{\mathsf{KRK}}$ the decryptor's public key, B forwards the public key generated by the IND-CPA$^{\mathrm{D}}$ game's challenger to $\mathcal{Z}'$.
- Whenever an honest client is told to encrypt a given message and outsource it to the calculator, B sets $m_0 = x_i$ (or $m_0 = y_j$), generates another message $m_1 \leftarrow M$ from the message space and sends both messages to the encryption oracle in order to receive a challenging ciphertext c. Then B forwards c to $\mathcal{Z}'$ as it came from the corresponding client. Lastly, B adds the new index $i + 1$ to the set of executed oracle requests $I = I \cup i + 1$ and sets $i := i + 1$
- all encrypted inputs given from $\mathcal{Z}'$ are treated as encryption oracle calls having $m_0 = m_1$, whose encryption is then sent to B. B then increases the oracle call index by $i = i + 1$.
- Whenever an initiator is told to start a computation round, B requests the evaluation oracle $\mathcal{O}_{\mathsf{EVAL}}$ with the indices of the encrypted messages by the clients $I' \subseteq I$ in order to receive a challenging ciphertext c. B then forwards c to $\mathcal{Z}'$ as coming from $\mathsf{S_C}$.
- Whenever a corrupted initiator is told to output a computation result, B gets the corresponding index j from the computation index list, and queries the decryption oracle $\mathcal{O}_{\mathsf{DEC}}$ with j in order to retrieve a plaintext result z_a. If

B gets $\perp$ from $\mathcal{O}_{\mathsf{DEC}}$, it queries the decryption oracle the initator's input and mask and computes the calculator's instructions on the plaintext, resulting in z'. Finally, B forwards z'_a to $\mathcal{Z}'$ as it came from $\mathsf{S_D}$.
- After the emulation, B outputs whatever $\mathcal{Z}'$ outputs.

Let us analyze B's advantage of guessing the right bit: $\mathcal{Z}'$'s probability of guessing the correct emulation mode is the same probability as B's probability in guessing the correct challenge bit. Thus, B wins the IND-CPA$^{\mathrm{D}}$ game with the same probability as the environment correctly distinguishes the real-world execution from the ideal-world simulation.

Now, let us assume that the hybrid games H_1 and H_2 are indistinguishable. Then we are able to construct a distinguisher $\mathcal{D}$ that is able to distinguish the real-world execution of the underlying protocol Π_{2PC} and its ideal-world simulation with non-neglible probability. The constructed $\mathcal{D}$ works as follows:

- Emulate the hybrid game H_1 with $\mathcal{Z}'$.
- Whenever a computation round is started, $\mathcal{D}$ gathers all (extracted or randomized) inputs and sends them to the protocol challenger.
- When $\mathcal{D}$ gets a communication transcript from the challenger, $\mathcal{D}$ computes the interim message $c_{d'} = c_S + c_{r,j}$ and forwards $c_{d'}$.
- When getting the message $(\mathsf{output?}, \mathsf{sid}, \mathsf{ssid})$, decrypt $c_{d'}$ to $d' = \mathsf{DEC}(\mathsf{sk}, c_{d'})$ and simulate receiving d' from $\mathsf{S_D}$.
- After the emulation B outputs whatever $\mathcal{Z}$ outputs.

$\mathcal{D}$ wins the game whenever it outputs the right guessing bit which is the same bit $\mathcal{Z}$ outputs when distinguishing H_2 from H_1. Thus, $\mathcal{D}$ distinguishes the real-world communication of Π_{2PC} from the simulated one with the same probability as $\mathcal{Z}$ successfully distinguishes H_{a+b+1} and H_{a+b}. Since $\mathcal{Z}$ successfully distinguishes both worlds with non-negligible probability, $\mathcal{D}$ is able to identify a simulated transcript with non-negligible probability.

A.5 Corrupted Decryptor (And Corrupted Initiators)

The simulation works as follows:

- In the server initialization phase, when getting a key registration message $(\mathsf{register}, \mathsf{sid}, \mathsf{sk}, \mathsf{pk})$ from $\mathsf{S_D}$, store the key pair and simulate forwarding it to $\mathcal{F}_{\mathsf{KRK}}$
- In the client registration phase, when getting a key retrieval request $(\mathsf{retrieve}, \mathsf{sid}, \mathsf{S_D})$ from $\mathsf{P_{Y,j}}$, take the stored pk and simulate receiving it from $\mathcal{F}_{\mathsf{KRK}}$.
- In the outsourcing phase of the initiator's inputs, when the simulator gets some encrypted messages from $\mathcal{Z}$, it decrypts the ciphertexts in order to extract the initiator's input and the initiator's mask, feeds the functionality with the extracted input and simulates sending it to the calculator.
- Whenever receiving the instruction $(\mathsf{start}, \mathsf{sid}, \mathsf{ssid})$ from $\mathcal{Z}$, forward the message to $\mathcal{F}_{\mathsf{OutComp}}$ and simulate forwarding the message to $\mathsf{S_C}$. When getting the result z from $\mathcal{F}_{\mathsf{OutComp}}$, follow the last step of the simulation of the underlying Π_{2PC} to get the last encrypted message c_d, compute $c_{d'} = c_d + c_{r,j}$

using the extracted masking value from the initiator's outsourcing phase and simulate receiving $c_{d'}$ from $\mathsf{S_C}$.
- Whenever getting the message $(\mathsf{ssid}, \mathsf{P_{Y,j}})$ from $\mathcal{F}_{\mathsf{OutComp}}$ for an honest $\mathsf{P_{Y,j}}$, draw a random message $d' \leftarrow \mathcal{M}$ from the message space, encrypt it to $c_{d'} = \mathsf{ENC}(\mathsf{pk}, d')$. Then store $(\mathsf{sid}, \mathsf{ssid}, \mathsf{P_{Y,j}}, d')$ and simulate receiving $c_{d'}$ from $\mathsf{S_C}$.
- In the output phase, whenever getting the request $(\mathsf{output?}, \mathsf{sid}, \mathsf{ssid})$ for a corrupted $\mathsf{P_{Y,j}}$ from $\mathcal{Z}$, output z received by $\mathcal{F}_{\mathsf{OutComp}}$.
- Whenever getting the message $(\mathit{output}, \mathsf{sid}, \mathsf{ssid}, \mathsf{P_{Y,j}})$ for an honest $\mathsf{P_{Y,j}}$ from $\mathcal{F}_{\mathsf{OutComp}}$, simulate receiving the message $(\mathit{output?}, \mathsf{sid}, \mathsf{ssid}, \mathsf{pid})$ from $\mathsf{P_{Y,j}}$, retrieve the stored interim result $(\mathsf{sid}, \mathsf{ssid}, \mathsf{P_{Y,j}}, d')$ and simulate sending $(\mathsf{sid}, \mathsf{ssid}, \mathsf{P_{Y,j}}, d')$ to $\mathsf{P_{Y,j}}$.

There are two general differences between the real-world execution and the ideal-world simulation:

1. For a corrupted initiator, the messages sent in the real-world execution from the calculator to the decryptor and from the decryptor to the initiator consist of the calculator's real-world computation of all clients' encrypted inputs and the decrypted and masked interim result d' whereas in the ideal-world simulation the messages consist of the two-party protocol's last simulated message added with the encrypted initiator's mask $c_{d'} = c_d + c_{r,j}$ using the corrupted initiator's input y_j and masking value r_j
2. For an honest initiator, the messages sent in the real-world execution from the calculator to the decryptor and from the decryptor to the initiator consist of the calculator's real-world computation of all clients' encrypted inputs and the masked interim result whereas in the ideal-world simulation the messages consist of a random value.

Therefore we have to differentiate three hybrid games:

- H_0: The real-world execution
- H_1: H_0 with the exception that in a computation and output phase initiated by a corrupted initiator, the messages are drawn accordingly to the simulation of Π_{2PC}
- H_2: The ideal world simulation

Let us begin with the indistinguishability between H_0 and H_1: Here we have to reduce the indistinguishability to the underlying standalone secure protocol Π_{2PC}. Therefore assume for the sake of contradiction that there exists an environment $\mathcal{Z}$ that is able to distinguish the real-world execution from the ideal-world simulation. Then we are able to create a non-interactive distinguisher $\mathcal{D}$ that is a able to distinguish a real-world execution of Π_{2PC} and ideal-world simulation with non-negligible probability by using $\mathcal{Z}$ in order to do the distinguishing work. The constructed $\mathcal{D}$ works as follows:

- Emulate a real-world execution of Π_{OutComp} with $\mathcal{Z}$.

- When starting the first computation round, gather all inputs from all input clients in order to compute $x = f_1(x_1, \cdots, x_n)$, the initiator's input and the key pair for the decryptor and feed the challenge protocol Π_{2PC} with the given inputs and the key pair $(\mathsf{sk}, \mathsf{pk})$ as R's auxiliary information and receive the generated transcript.
- When $\mathsf{S_D}$ has to receive an encrypted masked interim result from $\mathsf{S_C}$, take the last encrypted message c_S from the challenge transcript, compute $c_{d'} = c_S + c_{r,j}$ (where $c_{r,j}$ is the masking value generated by the initiator in the outsourcing phase) and letting $\mathsf{S_C}$ send $c_{d'}$ to $\mathsf{S_D}$. Then forward the round's result z to $\mathcal{Z}$.
- The other computation rounds are done as in the real-word execution.
- Output whatever $\mathcal{Z}$ outputs.

To evaluate $\mathcal{D}$'s advantage in distinguishing a simulated execution of Π_{2PC} from a real-world execution, let us look at the case where the transcript $\mathcal{D}$ received is the transcript of a real-world execution. In this case $\mathcal{D}$'s output is identically distributed as $\mathcal{Z}$'s output in the real-world execution.

Now let us look at the case where Π_{2PC}'s transcript is a simulated transcript. In this case the message $c_{d'} = c_S + c_{r,j}$ is generated as in H_1 and hence the interaction is the same as in H_1. Therefore $\mathcal{Z}$'s output is identically distributed as in H_1.

Following both cases, $\mathcal{D}$ outputs *ideal* with the same probability as $\mathcal{Z}$ does. Since we assume that $\mathcal{Z}$ is able to distinguish the real-world execution from the ideal-world simulation, it follows that $\mathcal{D}$ is able to distinguish a real protocol transcript from a simulated one with non-negligible probability, which contradicts the assumption that the real-world execution of Π_{2PC} is computationally indistinguishable from its simulation.

Next, we show the indistinguishability between H_1 and H_2: The only differences between H_1 and H_2 are the messages the corrupted decryptor gets from the honest calculator and the honest initiator gets from the corrupted decryptor. In the real-world execution these messages are the values the calculator evaluates over the encrypted masked inputs, which is a one-time pad whereas in the simulation those messages are the encryption of some randomly chosen messages from the message space. Since those values are perfectly indistinguishable, the execution of H_1 is perfectly indistinguishable from the execution of H_2.

A.6 Corrupted Decryptor and Corrupted Input Clients

- In the server initialization phase, when getting a key pair registration message $(\mathsf{register}, \mathsf{sid}, \mathsf{pk}, \mathsf{sk})$ from $\mathcal{Z}$, store the received key pair $(\mathsf{pk}, \mathsf{sk})$ and simulate receiving it from $\mathcal{F}_{\mathsf{KRK}}$.
- When a corrupted input client is told to outsource its input, extract the encrypted input and send the extracted input to $\mathcal{F}_{\mathsf{OutComp}}$ and simulate sending the encrypted inputs to the calculator.
- Whenever getting the message $(\mathsf{result}, \mathsf{ssid}, \mathsf{P_{Y,j}})$ from $\mathcal{F}_{\mathsf{OutComp}}$, draw a random value d', encrypt $c_{d'} = \mathsf{ENC}(\mathsf{pk}, d')$ and simulate receiving $c_{d'}$ from the $\mathsf{S_C}$.

– Whenever getting a message $(\mathsf{output}, \mathsf{ssid}, \mathsf{P}_{\mathsf{Y,j}})$ from $\mathcal{F}_{\mathsf{OutComp}}$, simulate sending d' to the $\mathsf{P}_{\mathsf{Y,j}}$.

The indistinguishability of the simulation is based on the information theoretic security of the initiator's masking value r_j, which is a one-time pad. Therefore the simulated message is the same as a real-world message. Following this, the real-world execution is perfectly indistinguishable from the ideal-world simulation.

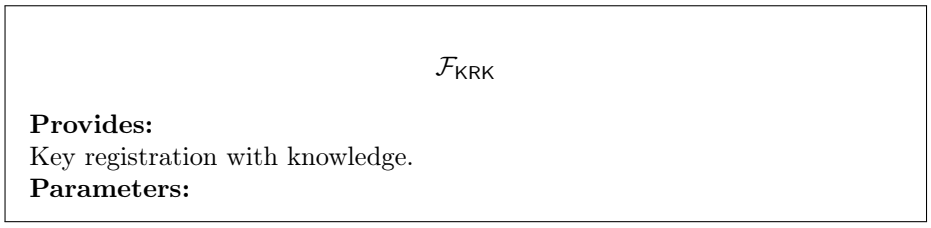
A.7 Corrupted Initiator, Subset of Input Clients and Decryptor

The constructed simulator works as follows:

– In the server initialization phase, when getting a key pair registration message $(\mathsf{register}, \mathsf{sid}, \mathsf{pk}, \mathsf{sk})$ from $\mathcal{Z}$, store the key pair and simulate forwarding it to $\mathcal{F}_{\mathsf{KRK}}$.
– In the client registration phase, whenever getting a public key request $(\mathsf{retrieve}, \mathsf{sid}, \mathsf{pid}\mathsf{S}_{\mathsf{D}})$ from $\mathcal{Z}$, take the stored pk and simulate receiving it from $\mathcal{F}_{\mathsf{KRK}}$.
– Whenever getting an outsourcing message $c_{x,i}$ from $\mathcal{Z}$, extract the input x_i from the ciphertext, send x_i to $\mathcal{F}_{\mathsf{OutComp}}$ and simulate sending $c_{x,i}$ to S_{C}.
– As the corrupted $\mathsf{P}_{\mathsf{Y,j}}$, when receiving the outsourcing message $(c_{y,j}, c_{r,j})$ from $\mathcal{Z}$, extract the initiator's input y_j and the mask r_j from the ciphertexts and send the input y_j to $\mathcal{F}_{\mathsf{OutComp}}$.
– When getting a message $(\mathsf{start}, \mathsf{ssid})$ from $\mathcal{Z}$, forward this message to $\mathcal{F}_{\mathsf{OutComp}}$ in order to receive the the result z of the current round and follow Π_{2PC}'s last simulation step of a corrupted receiver in order to get an encrypted message c_s. Then compute $c_{d'} = c_s + c_{r,j}$ and simulate receiving $c_{d'}$ from S_{C}.
– Result retrieval: When getting the message $(\mathsf{output?}, \mathsf{ssid})$ from $\mathcal{Z}$, simulate sending $c_{d'}$ to $\mathsf{P}_{\mathsf{Y,j}}$.

The indistinguishability of the simulation is based on on standalone security of the underlying protocol Π_{2PC}. To prove this statement we assume an environment $\mathcal{Z}$ that is able to distinguish the real-world protocol from the ideal-world simulation with non-negligible probability. The simulator is the exact simulator that is described in Appendix A.5 and also its security analysis is the same. Therefore we omit the reduction here and refer to Appendix A.5 for the full reduction proof.

B The Key Registration With Knowledge Functionality



– Function $\mathtt{f}_{\mathrm{Key}} : (\mathsf{sk}, \mathsf{pk}) \mapsto \begin{cases} \mathit{true}, & \text{well-formed key pair} \\ \mathit{false}, & \text{otherwise} \end{cases}$

State:

– Function $p_{\mathrm{Reg}} : \mathit{mid} \mapsto (P, \mathsf{sk}, \mathsf{pk})$ of pending registrations.
– Function $p_{\mathrm{Ret}} : \mathit{mid} \mapsto (P_i, P_j)$ of pending retrievals.
– Set $\mathbf{R}$ of registered tuples $(P, \mathsf{sk}, \mathsf{pk})$.

Behaviour:

– Upon receiving $(\mathtt{register}, \mathsf{sid}, \mathsf{sk}, \mathsf{pk})$ from a party P, draw fresh mid, send $(\mathtt{register}, \mathsf{sid}, \mathit{mid}, P, \mathsf{pk})$ to the adversary $\mathcal{A}$ and append $\mathit{mid} \mapsto (P, \mathsf{sk}, \mathsf{pk})$ to p_{Reg}.
– Upon receiving $(\mathtt{register\ ok}, \mathsf{sid}, \mathit{mid})$ from the adversary $\mathcal{A}$, retrieve $(P, \mathsf{sk}, \mathsf{pk}) := p_{\mathrm{Reg}}(\mathit{mid})$, check
  • $\mathtt{f}_{\mathrm{Key}}(\mathsf{sk}, \mathsf{pk}) = \mathit{true}$
  • $\nexists\ \mathsf{sk}', \mathsf{pk}' : (P, \mathsf{sk}', \mathsf{pk}') \in \mathbf{R}$
  • $\nexists\ P', \mathsf{sk}' : (P', \mathsf{sk}', \mathsf{pk}) \in \mathbf{R}$

  and append $(P, \mathsf{sk}, \mathsf{pk})$ to $\mathbf{R}$ if all checks were successful.
– Upon receiving $(\mathtt{retrieve}, \mathsf{sid}, P_i)$ from a party P_j, draw fresh mid, send $(\mathtt{retrieve}, \mathsf{sid}, \mathit{mid}, P_i, P_j)$ to the adversary $\mathcal{A}$ and append $\mathit{mid} \mapsto (P_i, P_j)$ to p_{Ret}.
– Upon receiving $(\mathtt{retrieve\ ok}, \mathsf{sid}, \mathit{mid})$ from the adversary $\mathcal{A}$, look up $(P_i, P_j) := p_{\mathrm{Ret}}(\mathit{mid})$ and $(P_i, \mathsf{sk}_i, \mathsf{pk}_i) \in R$. If no such entry exists in $\mathbf{R}$, set $pk_i := \perp$. Send $(\mathtt{retrieved}, \mathsf{sid}, \mathsf{pk}_i, P_i)$ to P_j.

Taken from the appendix in [11]. The three checks $\mathcal{F}_{\mathsf{KRK}}$ performs before finally registering a parties credentials guarantee that only valid key pairs may be registered, that each party registers at most one key pair and that no to parties can share the same public key.

C Two Party PSI Proof

Theorem 4 *Given an IND-CPA secure somewhat homomorphic encryption scheme* $\mathsf{HE} = (\mathsf{GEN}, \mathsf{ENC}, \mathsf{DEC})$*, the protocol* $\Pi_{\mathsf{2P-PSI}}$ *securely realizes* $\mathcal{F}_{\mathsf{2P-PSI}}$ *under sequential composition in the presence of a semi-honest adversary* $\mathcal{A}$*.*

Proof. To show the sequential composability of the protocol $\Pi_{\mathsf{2P-PSI}}$, it suffices to show that there exists a simulator $\mathcal{S}$ that simulated the interaction between the two parties without the presence of a distinguishing environment. Therefore $\mathcal{S}$ has to construct a transcript of the interaction between the two parties that is indistinguishable from the real-world interaction given the corrupted party's input and the functionality's output beforehand. Since we are only dealing with

passive adversaries, we don't have to rewind in case the adversary tries to deviate from the protocol.

Let us begin with the case that the sender is corrupted:

In this case the simulator $\mathcal{S}$ is given the input $\mathbf{x}$ and no output. Then the simulator works as follows:

- Generate a fresh key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GEN}(1^\lambda)$ using the key generation algorithm GEN. Then draw a random message $m' \leftarrow \mathcal{M}$ from HE's message space $\mathcal{M}$, encrypt $\mathbf{c_y} = \mathsf{ENC}(\mathsf{pk}, \mathbf{y})$ using the generated public key pk and simulate receiving the message $(\mathsf{pk}, \mathbf{c}_x)$ from the receiver R. Then draw a random value $\mathbf{r} \leftarrow \mathcal{M}$ from HE's message space $\mathcal{M}$. Then encrypt S's input $\mathbf{c_x} = \mathsf{ENC}(\mathsf{pk}, \mathbf{x})$ and the random value $\mathbf{c}_r = \mathsf{ENC}(\mathsf{pk}, \mathbf{r})$, compute $\mathbf{c_{z'}} = (\mathbf{c_y} - \mathbf{c_x}) \cdot \mathbf{c_r}$ and simulate sending $\mathbf{c_{z'}}$ to R.

The key pair is drawn in an honest manner. Therefore the simulated pk is perfectly indistinguishable from a real-world pk generated by the real-world honest sender. Although the simulated input does not equal S's actual input, its encryption is computationally indistinguishable from the real input's encryption thanks to the IND-CPA security of the homomorphic encryption scheme. Also, due to the indistinguishability of the ciphertexts, the simulated message sent to R is also compuationally indistinguishable from the encryption of a message sent by the real-world corrupted sender. Therefore the simulated transcript is computationally indistinguishable from the real-world execution.

Then let us continue with the case of a corrupted receiver:

In this case $\mathcal{S}$ gets the input $\mathbf{y}$ and the output $\mathbf{z}$ and has to generate a simulated interaction between the corrupted R and an honest S which is computationally indistinguishable from the real-world interaction. The simulator $\mathcal{S}$ works as follows:

- Generate a fresh key pair $(\mathsf{pk}, \mathsf{sk})$ and encrypt R's input $\mathbf{c_y} = \mathsf{ENC}(\mathsf{pk}, \mathbf{y})$ using the homomorphic encryption scheme HE and simulate sending the message $(\mathsf{pk}, \mathbf{c_y})$ to S.
- In the second simulation step, compute given the output $\mathbf{z}$ and R's input an interim result $\mathbf{z'}$ by setting $\mathbf{z}'_i = 0$ whenever for some $j \in [0, n]$ $\mathbf{y}_i = \mathbf{z}_j$ and fill the other indices with randomly drawn values. Then encrypt the result $\mathbf{c_{z'}} = \mathsf{ENC}(\mathsf{pk}, \mathbf{z'})$ and simulate receiving the message $\mathbf{c_{z'}}$.

Now we show that the simulated communication is statistically indistinguishable from a real-world protocol execution:

The first simulated message sent by the receiver is perfectly simulated since the simulator draws the key pair in an honest manner, just as real-world receiver would do. Also the encryption is done as in the real-world protocol. Therefore, there is no difference between the simulated first message and the real-world first message exchanged between R and S. In the second simulated message, the simulator doesn't get the honest S's input and therefore is not able to reconstruct the actual message shared between R and S. Yet, since the non-zero entries of $\mathbf{z'}$ sent in the real-world protocol have some randomness incorporated, those

entries are pseudorandom and therefore statistically indistinguishable from uniformly chosen random values, as done in the simulation. Therefore the simulated protocol is statistically indistinguishable from the real-world protocol execution.

D PSI Requirements

In the following, we will limit ourselves to only the most relevant requirements. Based on our interviews, we have identified the following security requirements.

- **SEC1**: Information Leakage – The server should not learn the criteria values (e. g., Role or Federal State)
- **SEC2**: Passive Attacker Model – We assume a passive attacker model, which means that we do not expect authorized clients and the service provider to act actively in a malicious way.
  - **SEC2.1**: We expect the service provider to implement all necessary measures to secure the service against external attackers, and to ensure that the client-side code—such as that executed in the browser—and all communication with the server are not maliciously manipulated by the provider. However, we consider the risk to be non-negligible that individuals at the provider with access to data storage may inspect the stored data, potentially misusing it for purposes not originally intended.
  - **SEC2.2**: We assume that clients do not maliciously manipulate their requests. The risk of clients violating this assumption is deemed negligible.
- **SEC3**: Clients/Server Collusion Resistance – Based on other requirements, including minimizing the usage threshold for all clients, we consider the risk of an insider attacker with access to server-stored data creating their own clients and inspecting their state to be non-negligible. Augmenting protections to meet SEC1 and SEC2 must take SEC3 into consideration.

In addition to the previously stated security requirements, we have functional requirements that will eliminate many of the protocols proposed so far in the literature on outsourced PSI. Obvious requirements such as correct matching of learning partners are omitted here.

- **FUNC1**: Client Activity – Clients should be able to go offline and come back online freely, without influencing the execution of the protocol.
- **FUNC2**: Failing Clients and Updates – Clients should be able to modify and eventually permanently delete their inputs without influencing the execution and correctness of the protocol.

Finally, there are non-functional requirements that we use to benchmark the final instantiated protocol.

- **NONFUNC1**: Server Storage
- **NONFUNC2**: Response Time for Matching requests

A centralized version where one trusted server is used to compute the intersections of the clients does not meet any of the security requirements stated above. Since we assume **SEC:2** as an attacker model, the attacker gaining access to the trusted server is able to read all data stored within the trusted server, especially the clients' datasets which are stored in plaintext. This obviously violates requirement **SEC:1**. Requirement **SEC:2.1** can be not met since even if all measurements against an external attacker are met, an internal attacker is still able to gain access to the datasets. This even still holds if there are protection measurements against malicious attacks, since eavesdropping attacks are usually not covered by those measures. Also, if we assume **SEC:2** as an attacker model, then the attacker gaining access to the trusted server has again all access to the plaintext datasets. If a client is able to corrupt the trusted server, then the adversary learns all outsourced datasets which is the most dangerous threat in our model. Therefore the model with a trusted server is not fitting for our case.

E Discussion on the Non-Collusion Assumption in the Real World

In the following, we discuss how attacks that violate the non-collusion assumption might appear in the real world and what protection methods could look like. We leave, however, the formal treatment of improving the robustness of our protocol against an attacker corrupting both $\mathsf{S_C}$ and $\mathsf{S_D}$ for future work. We will neither consider an outsider adversary that may penetrate the networks of the decryptor and calculator and thus violate the non-collusion assumption. Therefore, we focus on attacks that violate the non-collusion assumption from the perspective of an insider attacker corrupting different parties, which we deem as the most dangerous one to our protocol.

The first category refers to attacks involving organizations that operate $\mathsf{S_C}$ or $\mathsf{S_D}$ and knowingly collude with each other—often under the instruction of a C-level executive—to share data. The motivation for this level of corruption can vary, as described in detail for the public sector by [31]. From our perspective, the most important factor driving such collusion is the potential cost to the organizations if the collusion is uncovered.

Therefore, we propose examining the potential minimal costs that organizations operating $\mathsf{S_C}$ or $\mathsf{S_D}$ may incur if their collusion is uncovered, in order to assess the risk of violating the non-collusion assumption due to organizational-level corruption.

If we exclude organizational-level corruption, the only remaining threat to the non-collusion assumption comes from individual insiders who have access to the data being processed by $\mathsf{S_C}$ and $\mathsf{S_D}$. The motivations of such individuals can also be diverse. In our private set intersection (PSI) instantiation, we identified a particularly realistic threat that underpins our adversary model. This is because our application of the PSI protocol was aimed at matching learning partners, which include underage people. This implies a direct risk of data misuse by individuals within the operating organization who have access to this sensitive

information. Although our honest-but-curious adversary model accounts for such attackers within $\mathsf{S_C}$ or $\mathsf{S_D}$, the guarantees break down when these adversaries collude with each other. In this specific use case, we struggle to identify a scenario in which two distinct individuals working for different operators at $\mathsf{S_C}$ and $\mathsf{S_D}$ might collude—without knowing each other beforehand-—to misuse the data of underage individuals for personal gain. Furthermore, even if such individuals knew each other prior to the protocol's execution, it is difficult to imagine a realistic scenario where they would deliberately take jobs at $\mathsf{S_C}$ and $\mathsf{S_D}$ with the intent of colluding in a subsequent step to misuse data concerning underage children. Moreover, we identified that the reasoning and threat modeling of this kind of collusion is not possible in general and it strongly depends on the exact organizations and its employees operating $\mathsf{S_C}$ and $\mathsf{S_D}$.

Ultimately, there remains the following question: What countermeasures can be deployed to prevent an attack that cannot be fully described in general terms? We propose to first exam the requirements of such an attack and then implement countermeasures that can thwart those specific requirements.

A key requirement is the possibility of a passive attack on $\mathsf{S_C}$ and $\mathsf{S_D}$ on the individual level. While our protocol protects against information leakage, additional countermeasures—some of which may be organizational in nature—could be implemented to further prevent this type of attack. It is important to distinguish between protecting against information leakage from an attack and preventing or tracing the attack itself. Preventive measures typically involve authentication or authorization mechanisms, while tracing measures require secure logging and effective monitoring of those logs.

Since these measures are highly dependent on the specific circumstances of the operators of $\mathsf{S_C}$ and $\mathsf{S_D}$, a more detailed discussion falls outside the scope of this work.

F Choice of Authentication, Authorization and the Passive Adversary Model of Π_{OutComp}

When deploying an instantiation in the real world, one will eventually face the question of how to authorize the input clients $\mathsf{P_{X,i}}$ and the initiating clients $\mathsf{P_{Y,j}}$. This issue is not specific to Π_{OutComp} but applies to any protocol that relies on the passive adversary model. Without proper authentication and authorization mechanisms, the system would be vulnerable to trivial denial-of-service attacks by spamming encrypted inputs. The protocol Π_{OutComp} does not include any such protections, and it is implicitly assumed—based on the chosen adversary model and party setup—-that every input client and initiator participating in the protocol is authorized and trusted to behave passively (if being corrupted). This implies that authorization is presumed to occur when a party is granted access to communicate with $\mathsf{S_C}$ and $\mathsf{S_D}$. At first glance, this might seem as an unrealistic choice of an adversary model, but there is, from our point of view, a sound reasoning behind it.

Consider, for example, a simple survey aggregation scenario, which is one of the most straightforward instantiations of Π_{OutComp}. Imagine the participation link (i. e., communication access with S_C) is made publicly available. In such a case, many people might attempt to cause disruption by submitting contrived responses to skew the overall results. We remind the reader that a similar public-access attempt was made by Microsoft with the chatbot Tay. Microsoft had to shut down the project within 24 hours because Tay began generating inappropriate content, as discussed by Wolf et al. [60]. In contrast, many other chatbots until today have not exhibited similar inappropriate behavior. The key difference was that Tay was trained in a public setting, while the other chatbots were trained in authorized, controlled environments. In the latter setting, we can reasonably assume that the training data was not intentionally manipulated with malicious intent. This reasoning also applies to the passive adversary model: while the security of Π_{OutComp} would be easily compromised in a public setting, in an authorized environment, we are able to thwart any remaining honest-but-curious attacks, which is a typical choice for protocols proven secure within passive adversary models.

The main takeaway is that Π_{OutComp} requires proper authentication and authorization for participating clients to ensure that only honest-but-curious clients are allowed to participate. There are plenty of methods available for this, making the choice of methods a relatively minor issue. The bigger challenge is to ensure that participants must be trusted to act in a honest-but-curious way only. If the latter challenge cannot be addressed with negligible risk, then the application intended to be realized by an instantiation of Π_{OutComp} will not work as intended.