# There Is Always an Exception: Controlling Partial Information Leakage in Secure Computation[*]

Máté Horváth Levente Buttyán Gábor Székely Dóra Neubrandt

Budapest University of Technology and Economics
Laboratory of Cryptography and System Security (CrySyS Lab)
{mhorvath,buttyan,gszekely,dneubrandt}@crysys.hu

November 11, 2019

**Abstract**

Private Function Evaluation (PFE) enables two parties to jointly execute a computation such that one of them provides the input while the other chooses the function to compute. According to the traditional security requirements, a PFE protocol should leak no more information, neither about the function nor the input, than what is revealed by the output of the computation. Existing PFE protocols inherently restrict the scope of computable functions to a certain function class with given output size, thus ruling out the direct evaluation of such problematic functions as the identity map, which would entirely undermine the input privacy requirement. We observe that when not only the input $x$ is confidential but certain partial information $g(x)$ of it as well, standard PFE fails to provide meaningful input privacy if $g$ and the function $f$ to be computed fall into the same function class.

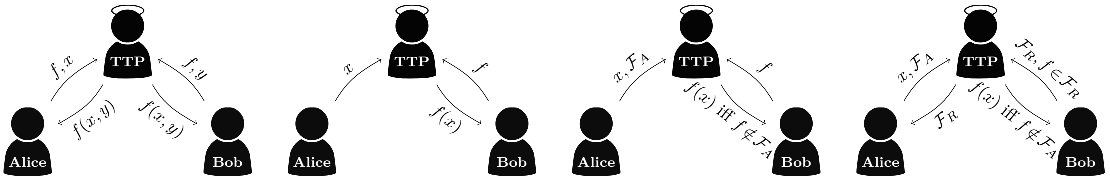
Our work investigates the question whether it is possible to achieve a reasonable level of input and function privacy simultaneously even in the above cases. We propose the notion of Controlled PFE (CPFE) with different flavours of security and answer the question affirmatively by showing simple, generic realizations of the new notions. Our main construction, based on functional encryption (FE), also enjoys strong reusability properties enabling, e.g. fast computation of the same function on different inputs. To demonstrate the applicability of our approach, we show a concrete instantiation of the FE-based protocol for inner product computation that enables secure statistical analysis (and more) under the standard Decisional Diffie–Hellman assumption.

**Keywords**: Cryptographic Protocols, Private Function Evaluation, Functional Encryption, Oblivious Transfer, Secure Data Markets

## 1 Introduction

Secure two-party computation (2PC) a.k.a. secure function evaluation (SFE) protocols enable two parties, Alice and Bob, to compute a function of their choice on their private inputs without disclosing their secrets to each other or anyone else (see Fig. 1a). In real life, however, the participants not necessarily have interchangeable roles. We call private function evaluation (PFE) a protocol if one party can alone choose the function to evaluate, while the other provides the input to it (see Fig. 1b) while both of them intends to hide their contribution. PFE can be realized by invoking 2PC after the function was turned into data. A universal function [Val76] is a "programmable function" that can implement *any* computation up to a given complexity. It takes

[*] This is the full version of [HBSN19]. In case of citing our work, please cite the proceedings version.



(a) SFE or 2PC (b) PFE (c) CPFE (d) rCPFE

Figure 1: Comparison of the *ideal functionality* of different concepts for secure function evaluation, realized with the help of a trusted third party (TTP). The key difference lies in which information Alice and Bob can or cannot have access to.

two inputs, the description of the function to be computed and the input to it. By evaluating a public universal function using 2PC, all feasibility results extend from 2PC to PFE. Improving efficiency turns out to be more challenging. Indeed, universal functions cause significant – for complex computations even prohibitive – overhead, and the elimination of this limitation was the primary focus of PFE research [KS08, KS16].

In this work, we initiate the study of a security issue that – to the best of our knowledge – received no attention earlier. More concretely, we focus on the opportunities of the input provider to control the information leakage of her input. As PFE guarantees Bob that his function is hidden from Alice, he can learn some information about the input of Alice such that it remains hidden what was exactly revealed. Disclosing the entire input by evaluating the identity function is typically ruled out by the restriction that the computable function class has shorter output length than input length. At the same time, the following question arises: is it really possible to determine the computable function class so that no function is included which could reveal sensitive information about the input? We argue that most often exceptions occur in every function class, so measures are required to also protect such partial information besides the protection of the input as a whole. As intentional partial information recovery does not cause anomalies when only the function provider, Bob receives the function's output, later on we consider this scenario.

For a simple and illustrative example, let us recall one of the most popular motivating applications for PFE. In privacy-preserving credit checking [PSS09, §7], Alice feeds her private data to a Boolean function of her bank (or another service provider) that decides whether she is eligible for credit or not. Using PFE for such computation allows Alice to keep her data secret and the bank to hide its crediting policy. Notice that the function provider can extract *any binary information* about the input and use it, e.g. to discriminate clients. The leaked partial information can be, e.g. gender or the actual value of any indicator variable about the data that should not be necessary to reveal for credit checking. Our goal is to enable Alice to rule out the leakage of specific sensitive information in PFE without exposing what partial information she wants to hide.

## 1.1 Our Contributions

Our contributions can be summarized as follows.

- We initiate the study of partial information protection in the context of private function evaluation.
- To take the first step, we put forward the notion of Controlled PFE (CPFE) and formally define its security (see Fig. 1c for its ideal functionality). We also devise a relaxed definition, called rCPFE (see Fig. 1d) that guarantees weaker (but still reasonable) $k$-anonymity style function privacy leading to a trade-off between security and efficiency.
- Then we show conceptually simple, generic realizations of both CPFE and rCPFE. In the latter case, we utilize the modified function privacy guarantee (through using functional encryption) to enable the reusability of the protocol messages in case of multiple function

evaluations. As a result, in our rCPFE when evaluating the same function(s) on multiple, say $d$ inputs, the communication and online computation overhead only increases with an additive factor proportional to $d$ instead of a multiplicative factor as in ordinary PFE.

- To demonstrate the practicality of the rCPFE approach, we instantiate our generic protocol for the inner product functionality enabling secure statistical analysis in a controlled manner under the standard Decisional Diffie–Hellman (DDH) assumption. Our proof of concept implementation shows that the reusability property indeed results in a significant performance improvement over the state of the art secure inner product evaluation method [DSZ15].

## 1.2 Applications

We believe that in most PFE applications, the evaluated function class also permits the leakage of potentially sensitive partial information about the input as our above example demonstrates this even for very restricted Boolean functions. To motivate our inner product rCPFE, we mention two of its possible application scenarios.

**Logistic Regression Evaluation.** The linear part of logistic regression computation is an inner product of the input and weight vectors. Our inner product rCPFE can help to rule out weight vectors that are unlikely to belong to a model but are base vectors that could reveal a sensitive input vector element.

**Location Privacy.** Let us assume that a "data broker" (DB) periodically collects location-based information in vector form, where vector elements correspond to information related to specific positions. Such data can be important for service providers (SP), offering location-based services, without the proper infrastructure to collect the necessary data. During their interaction that can be an inner product computation,[1] the SP should hide the location of its users, while the DB may want to protect the exact information in specific locations or to adjust higher price if specific measurements are used. These can be achieved by having control over the possible queries of SP.

## 1.3 Related Work

Some PFE variants share ideas with our concepts. Semi-private function evaluation (semi-PFE) [PSS09, KKW17] for instance, also relaxes the function privacy requirement of PFE by revealing the topology of the function being evaluated. While this relaxation also leads to a useful trade-off between function privacy and efficiency, unfortunately, the available extra information about the function does not necessarily allow Alice to rule out the evaluation of functions that are against her interest.

Selective private function evaluation (SPFE) [CIK$^+$01] deals with a problem that is orthogonal to the one considered in this paper. Namely, SPFE also aims to conceal information that is leaked in PFE. However, instead of protecting Alice (the data owner), it intends to increase the security of Bob by hiding from Alice the location of the function's input in her database via using private information retrieval (PIR).

Leaving the field of PFE and comparing our work to related problems in secure computation, we find that hiding the computed function raises similar issues in other contexts. [BGJS16] put forth the notion of verifiable obfuscation that is motivated by the natural fear for executing unknown programs. The goal here is similar than in our setting: some assurance is required that the hidden functionality cannot be arbitrary. However, the fundamental difference between our CPFE and the verifiable obfuscation and verifiable FE of [BGJS16] is that while the latter ones enforce correctness when an obfuscator or authority may be dishonest, CPFE tries to disable semi-honest parties to evaluate specific functions (i.e. to handle exceptions in PFE).

[1] E.g. multiplying the data vector with a position vector (that is non-zero in all positions representing locations close to the user – possibly containing weights depending on the distance – and zero otherwise) can give useful information.

The functionality is parametrized by two integers $k < n$, and two parties: a sender $\mathcal{S}$ and a receiver $\mathcal{R}$.

FUNCTIONALITY:
On input $m_1, \dots, m_n$ messages from $\mathcal{S}$ and an index set $\{i_1, \dots, i_k\} \subset [n]$ from $\mathcal{R}$

- $\mathcal{S}$ obtains no output,
- $\mathcal{R}$ receives $m_{i_1}, \dots, m_{i_k}$ but nothing else.

Figure 2: Ideal functionality $\mathcal{F}_{OT^n_k}$ of $k$ out of $n$ OT.

Our rCPFE is built upon functional encryption (FE) in a black-box manner. This generalization of traditional encryption was first formalized by [BSW11]. While general-purpose FE candidates [GGH$^+$13, GGHZ16] currently rely on untested assumptions like the existence of indistinguishability obfuscation or multilinear maps, our application does not require such heavy hammers of cryptography (see details in §2.2). In the context of FE, [NAP$^+$14] raised the question of controllability of function evaluation. The essential difference, compared to our goals, is that they want to limit repeated evaluations of the *same* function[2] that they solve with the involvement of a third party.

Finally, we sum up the state of the art of private inner product evaluation. The provably secure solutions are built either on partially homomorphic encryption schemes [GLLM04, DC14] or 2PC protocols [DSZ15] but public-key inner product FE [ABCP15] is also capable of the same task. At the same time, several ad-hoc protocols achieve better performance in exchange for some information leakage (see, e.g. [ZWH$^+$15] and the references therein), but these constructions lack any formal security argument.

# 2 Preliminaries

In this section, we briefly summarize the relevant background for the rest of the paper. We will always assume that the participants of the considered protocols are semi-honest, i.e. while following the protocol honestly, they try to recover as much information from the interactions as they can. We also use the OT-hybrid model that assumes that the parties have access to an ideal process that securely realizes oblivious transfer, which we discuss in more detail in §2.1.

## 2.1 Oblivious Transfer

Oblivious transfer (OT) is one of the most fundamental primitives in cryptography and a cornerstone of secure computation. It enables transferring data between two parties, the sender ($\mathcal{S}$) and the receiver ($\mathcal{R}$, a.k.a. chooser), in a way that protects both of them. $\mathcal{S}$ can be sure that $\mathcal{R}$ only obtains a subset of the sent messages, while $\mathcal{R}$ is assured that $\mathcal{S}$ does not know which messages he selected to reveal. In Fig. 2 the ideal functionality of $k$ out of $n$ OT [CT05] is represented that we are also going to rely on.

While being a public-key primitive, so-called OT-extension protocols enable rather efficient OT evaluation. To do so, the participants first pre-compute a limited number of "base-OTs" with certain inputs that are independent of their real inputs. Then using the obtained values, they can evaluate a much larger number of OTs by executing more efficient symmetric-key operations only. This kind of efficiency improvement automatically applies to our protocols after substituting plain OT, with OT-extension with the same functionality [KKRT16, RR17].

[2] In FE schemes, the control over the computable functions is in the hand of the master secret key holder, so this is not an issue unlike in PFE.

## 2.2 Functional Encryption

As we already introduced, FE is a generalized encryption scheme that enables certain computations on hidden data for authorized parties. Both public- and secret-key variants are known, but here we limit ourselves to the secret-key setting that suffices for our purposes. An sk-FE scheme consists of the following four algorithms.

$\mathsf{FE.Setup}(\lambda) \to (\mathsf{msk_{FE}}, \mathsf{pp_{FE}})$ Upon receiving a security parameter $\lambda$ it produces the public system parameters $\mathsf{pp_{FE}}$ and the master secret key $\mathsf{msk_{FE}}$.

$\mathsf{FE.Enc}(\mathsf{msk_{FE}}, x) \to \mathsf{ct}$ The encryption algorithm takes the master secret key $\mathsf{msk_{FE}}$ and a message $x$ and outputs a ciphertext $\mathsf{ct}$.

$\mathsf{FE.KeyGen}(\mathsf{msk_{FE}}, f) \to \mathsf{fsk}_f$ The key generation algorithm can be used to generate a functional secret key $\mathsf{fsk}_f$ for a function $f$ with the help of the $\mathsf{msk_{FE}}$.

$\mathsf{FE.Dec}(\mathsf{ct}, \mathsf{fsk}_f) \to y$ Having a functional secret key $\mathsf{fsk}_f$ (for function $f$) and a ciphertext $\mathsf{ct}$ (corresponding to $x$), the decryption outputs the value $y$.

The correctness of FE requires that if $\mathsf{fsk}_f$ and $\mathsf{ct}$ were indeed generated with the corresponding algorithms using inputs $f$ and $x$ respectively, then $y = f(x)$ must hold. Regarding security, in this work we are going to use the non-adaptive simulation-based security definition of FE [GVW12], which we recall in Appendix A. We note that while the SIM security of FE is impossible to realize in general [BSW11], for several restricted – yet important – cases it is still achievable, e.g. when the number of functional keys are a priori bounded [GVW12], or when the computable function class is restricted [ALS16]. As our applications also use these restrictions, known FE impossibility results do not affect the way we use FE.

# 3 General Approaches for Securing Partial Input Information in PFE

In this part, we introduce the notion of controlled PFE and in §3.1 formally define its security in different flavours. Next, in §3.2–3.3, we propose two general protocols satisfying these security requirements.

## 3.1 Definitional Framework

Our first security definition for controlled PFE captures the intuitive goal of extending the PFE functionality with a blind function verification step by $P_1$ to prevent unwanted information leakage. See the corresponding ideal functionality $\mathcal{F}_{\mathrm{CPFE}}$ in Fig. 3 that we call controlled PFE, and the security definition below. For the ease of exposition, later on we denote the inputs of the participants as $\mathsf{inp} = (\{x_i\}_{i\in[d]}, \mathcal{F}_A, \{f_j\}_{j\in[k]})$ with the corresponding parameters.

**Definition 1** (SIM security of CPFE wrt. semi-honest adversaries)**.** *Let* $\Pi$ *denote a Controlled PFE (CPFE) protocol for a function class* $\mathcal{F}$ *with functionality* $\mathcal{F}_{\mathrm{CPFE}}$ *(according to Fig. 3). We say that* $\Pi$ *achieves SIM security against semi-honest adversaries, if the following criteria hold.*

- Correctness: *the output computed by* $\Pi$ *is the required output, i.e.*

$$\Pr[\mathsf{output}^{\Pi}(1^\lambda, \mathsf{inp}) \neq \mathcal{F}_{\mathrm{CPFE}}(\mathsf{inp})] \leq \mathsf{negl}(\lambda).$$

- Function Privacy: *there exists a probabilistic polynomial time (PPT) simulator* $\mathcal{S}_{P_1}$*, s.t.*

$$\{\mathcal{S}_{P_1}(1^\lambda, \{x_i\}_{i\in[d]}, \mathcal{F}_A)\}_{\lambda, x_i, \mathcal{F}_A} \overset{c}{\approx} \{\mathsf{view}^{\Pi}_{P_1}(1^\lambda, \mathsf{inp})\}_{\lambda, x_i, f_j, \mathcal{F}_A}.$$

- Data Privacy: *there exists a PPT simulator* $\mathcal{S}_{P_2}$*, s.t.*

$$\{\mathcal{S}_{P_2}(1^\lambda, \{f_j\}_{j\in[k]}, \{y'_{i,j}\}_{i\in[d], j\in[k]}\}_{\lambda, f_j} \overset{c}{\approx} \{\mathsf{view}^{\Pi}_{P_2}(1^\lambda, \mathsf{inp})\}_{\lambda, x_i, f_j, \mathcal{F}_A}$$

PARAMETERS: participants $P_1, P_2$, a class $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$ of deterministic functions *[and an integer $\kappa > k$]*

FUNCTIONALITY:
On inputs $x_1, \dots, x_d \in \mathcal{X}$ and $\mathcal{F}_A \subset \mathcal{F}$ from $P_1$; and $\mathcal{F}_B = \{f_1, \dots, f_k\} \subset \mathcal{F}$ from $P_2$

- $P_1$ receives no output, *[or $P_1$ receives $\mathcal{F}_R$ s.t. $\mathcal{F}_B \subset \mathcal{F}_R \subset \mathcal{F}$ and $|\mathcal{F}_R| = \kappa$]*
- $P_2$ obtains $\{y'_{i,j} = f'_j(x_i)\}_{i \in [d], j \in [k]} \subset \mathcal{Y} \cup \{\perp\}$ for

$$f'_j(x_i) = \begin{cases} f_j(x_i) & \text{if } f_j \notin \mathcal{F}_A \\ \perp & \text{otherwise.} \end{cases}$$

Figure 3: Ideal functionalities for $\mathcal{F}_{\mathrm{CPFE}}$ and $\mathcal{F}_{\mathrm{rCPFE}}$ (see the extensions in brackets) formulated generally for multiple inputs and multiple functions.

*where* $\mathsf{inp} = (\{x_i\}_{i \in [d]}, \mathcal{F}_A, \{f_j\}_{j \in [k]}), f_j \in \mathcal{F}, \mathcal{F}_A \subset \mathcal{F}, x_i \in \mathcal{X}, y'_{i,j} \in \mathcal{Y} \cup \{\perp\}$, *and* $\lambda \in \mathbb{N}$.

We also propose a relaxation of Def. 1, which on the one hand gives up perfect function privacy but on the other, allows us to construct efficient protocols while still maintaining a $k$-anonymity style guarantee for function privacy. As SIM security alone cannot measure how much information is leaked by a set of functions, we formulate an additional requirement to precisely characterise function privacy.

**Definition 2** (SIM security of relaxed CPFE wrt. semi-honest adversaries)**.** *Let* $\Pi$ *denote a relaxed CPFE (rCPFE) protocol for a function class* $\mathcal{F}$ *with functionality* $\mathcal{F}_{\mathrm{rCPFE}}$ *(according to Fig. 3). We say that* $\Pi$ *achieves SIM security against semi-honest adversaries, if the following criteria hold.*

- Correctness: *the output computed by* $\Pi$ *is the required output, i.e.*

$$\Pr[\mathsf{output}^{\Pi}(\lambda, \kappa, \mathsf{inp}) \neq \mathcal{F}_{\mathrm{rCPFE}}(\kappa, \mathsf{inp})] \leq \mathsf{negl}(\lambda).$$

- Function Privacy: *is defined in two flavours:*
  - $\kappa$*-relaxed function privacy holds, if* $\exists\, \mathcal{S}_{P_1}$*, a PPT simulator, s.t.*

$$\{\mathcal{S}_{P_1}(1^{\lambda}, \kappa, \{x_i\}_{i \in [d]}, \mathcal{F}_A)\}_{\lambda, \kappa, x_i, \mathcal{F}_A} \overset{c}{\approx} \{\mathsf{view}^{\Pi}_{P_1}(1^{\lambda}, \kappa, \mathsf{inp})\}_{\lambda, \kappa, x_i, f_j, \mathcal{F}_A}.$$

  - *Strong* $\kappa$*-relaxed function privacy holds if besides the existence of the above* $\mathcal{S}_{P_1}$*, it also holds that for any PPT* $\mathcal{A}$*:*

$$\left| \Pr[\mathcal{A}(\mathsf{aux}, \mathcal{F}_R) \in \mathcal{F}_B] - \frac{k}{\kappa} \right| \leq \mathsf{negl}(\lambda)$$

*where* $\mathsf{aux} \in \{0,1\}^*$ *denotes some a priori known auxiliary information about* $\mathcal{F}_B$.

- Data Privacy: *there exists a PPT simulator* $\mathcal{S}_{P_2}$*, s.t.*

$$\{\mathcal{S}_{P_2}(\lambda, \kappa, \{f_j\}_{j \in [k]}, \{y'_{i,j}\}_{i \in [d], j \in [k]}\}_{\lambda, \kappa, f_j} \overset{c}{\approx} \{\mathsf{view}^{\Pi}_{P_2}(\lambda, \kappa, \mathsf{inp})\}_{\lambda, \kappa, x_i, f_j, \mathcal{F}_A}$$

*where* $\mathsf{inp} = (\{x_i\}_{i \in [d]}, \mathcal{F}_A, \{f_j\}_{j \in [k]}), f_j \in \mathcal{F}, \mathcal{F}_A \subset \mathcal{F}, x_i \in \mathcal{X}, y'_{i,j} \in \mathcal{Y} \cup \{\perp\}$, *and* $\lambda, \kappa \in \mathbb{N}$.

Protocol $\Pi_{\mathcal{F}}^{\text{CPFE}}$

Parameters: $\lambda$ parametrizing security, a function class $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$, and a universal circuit $UC$ for the function class $\mathcal{F}$

Inputs:

- $P_1$: $x, \mathcal{F}_A \subset \mathcal{F}$
- $P_2$: $f \in \mathcal{F}$

Protocol:

Using a secure two-party computation protocol, $P_1$ and $P_2$ executes the following computation on their inputs:

- If $f \in \mathcal{F}_A$, return $\perp$ to both $P_1$ and $P_2$.
- Otherwise compute the universal circuit $UC(f, x) = f(x) \in \mathcal{Y}$ outputting $\perp$ to $P_1$ and $f(x)$ to $P_2$.

Figure 4: General 2PC-based CPFE

## 3.2 Universal Circuit-based CPFE

The natural approach for realizing CPFE comes from the traditional way of combining universal circuits and SFE to obtain PFE. Fig. 4 shows how the same idea with conditional evaluation leads to CPFE in the single input, single function setting. The following theorem is a straightforward consequence of the security of SFE.

**Theorem 1.** *The CPFE protocol of Fig. 4 is secure according to Def. 1, if the used SFE protocol is SIM secure in the semi-honest model.*

The main drawback of this approach is that when extending the protocol to handle multiple inputs or functions, its complexity will multiplicatively depend on the number of inputs or functions because of the single-use nature of 2PC.

## 3.3 Reusable Relaxed CPFE from FE

We observe that the notion of rCPFE not only allows the input provider to verify the functions to be evaluated but also opens the door for making parts of the protocol messages reusable multiple times, thus leading to significant efficiency improvements.

A naive first attempt to realize rCPFE is to execute the computation on the side of $P_1$. Upon receiving a $\kappa$ function descriptions (including both the intended and dummy functions) $P_1$ can easily verify the request and evaluate the allowed ones on her input. The results then can be shared with $P_2$, using an OT scheme achieving both the required data and function privacy level. Unfortunately, the $\kappa$ function evaluations lead to scalability issues. The subsequent natural idea is to shift the task of function evaluation to $P_2$, to eliminate the unnecessary computations and to hide the output from $P_1$ entirely. Since at this point $P_1$ has both the inputs and the functions to evaluate, the task resembles secure outsourcing of computation where function evaluation must be under the strict control of $P_1$. These observations lead us to the usage of FE and the protocol in Fig. 5 in which both ciphertext and functional keys can be reused in multiple computations. When instantiated with the FE scheme of [GVW12], $\Pi_{\mathcal{F}}^{\text{rCPFE}}$ can be used for all polynomial sized functions in theory (in practice verifying the circuits would be a bottleneck).

**Theorem 2.** *The protocol of Fig. 5 is SIM secure according to Def. 2 achieving $\kappa$-relaxed function privacy for $k$ function queries by $P_2$, if the underlying FE scheme is $k$-query non-adaptive SIM secure ($k$-NA-SIM) for a single message and the used OT protocol is SIM secure against semi-honest adversaries.*

Protocol $\underline{\Pi_{\mathcal{F}}^{\text{rCPFE}}}$

Parameters: $\kappa, \lambda$ parametrizing security and function class $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$

Inputs:

- $P_1$: $x_1, \dots, x_d \in \mathcal{X}, \mathcal{F}_A \subset \mathcal{F}$
- $P_2$: $\mathcal{F}_B = \{f_1, \dots, f_k\} \subset \mathcal{F}$

Protocol:

Online Phase

**Step I.** To initiate the evaluation of functions in $\mathcal{F}_B$, $P_2$

(1) samples $\kappa - k$ functions randomly: $\{f_i \leftarrow_{\$} \mathcal{F}\}_{k<i\leq\kappa}$,

(2) takes a random permutation on $\kappa$ elements to set $\mathcal{F}_R := (\hat{f}_1, \dots, \hat{f}_\kappa)$, where $\hat{f}_i = f_{\sigma^{-1}(i)}$ so that each $f_i$ ends up at position $\sigma(i)$ in the sequence,

(3) finally, sends[3]$\mathcal{F}_R$ to $P_1$.

**Step II.** Upon receiving a function request $\mathcal{F}_R$, $P_1$

(1) samples $(\mathsf{msk}_{\mathsf{FE}}, \mathsf{pp}_{\mathsf{FE}}) \leftarrow_{\$} \mathsf{FE.Setup}(\lambda)$,

(2) encrypts the input data: $\mathsf{ct}_j \leftarrow_{\$} \mathsf{FE.Enc}(\mathsf{pp}_{\mathsf{FE}}, \mathsf{msk}_{\mathsf{FE}}, x_j)$ for all $j \in [d]$,

(3) determines the index set of allowed functions $I := \{i \mid \hat{f}_i \notin \mathcal{F}_A\}$,

(4) generate functional keys $\mathsf{fsk}_{\hat{f}_i} \leftarrow_{\$} \mathsf{FE.KeyGen}(\mathsf{pp}_{\mathsf{FE}}, \mathsf{msk}_{\mathsf{FE}}, \hat{f}_i)$ for all $i \in I$.

(5) finally, sends $\mathsf{pp}_{\mathsf{FE}}$ and $\{\mathsf{ct}_j\}_{j\in[d]}$ to $P_2$.

**Step III.** $P_1$ and $P_2$ invoke the $\mathcal{F}_{OT^n_k}$-functionality:

(1) $P_1$ act as *sender* with $\kappa$ messages as input: $m_i = \mathsf{fsk}_{\hat{f}_i}$ for $i \in I$ and $m_i = \perp$ for $i \in [\kappa] \setminus I$.

(2) $P_2$ act as *receiver* with input $(\sigma(1), \dots, \sigma(k))$

(3) $P_2$ receives $m_{\sigma(1)}, \dots, m_{\sigma(k)}$ where $m_{\sigma(i)} = \mathsf{fsk}_{f_i}$ or $m_{\sigma(i)} = \perp$ if it was not an allowed function (thus implicitly also obtaining the index set $I \cap [k]$).

Offline Phase

$P_2$ can evaluate the allowed functions from $\mathcal{F}_B$ on all input of $P_1$ by running $\mathsf{FE.Dec}(\mathsf{fsk}_{f_i}, \mathsf{ct}_j) = f_i(x_j)$ for all $i \in I \cap [k]$.

Figure 5: General rCPFE construction.

The proof of the theorem is postponed to Appendix B.

**Corollary 1.** *The protocol of Fig. 5 also achieves* strong *$\kappa$-relaxed function privacy if in (1) of Step I., all $f_i$ are sampled from the same distribution as the elements of $\mathcal{F}_B$ and* $\mathsf{aux} = \perp$.

## 4 Concrete Instantiation for Inner Products

To demonstrate the practicality of our approach, we instantiate our generic rCPFE protocol (Fig. 5) using the $k$-NA-SIM secure FE scheme of [ALS16] for the inner product functionality and the semi-honest 1 out of $\kappa$ OT protocol of [Tze04]. Theorem 2 and the assumptions of [ALS16, Tze04] directly imply the following theorem.

[3] Depending on $\mathcal{F}$ and the sampling of the dummy functions, communication cost of transferring the function descriptions can be reduced. In Appendix C.3 we describe such optimizations for the inner product function class.

**Theorem 3.** *There is a SIM secure* rCPFE *protocol (according to Def. 2) for inner product computation, achieving $\kappa$-relaxed function privacy, if the DDH assumption holds.*

**Corollary 2.** *The inner product* rCPFE *protocol derived from $\Pi_{\mathcal{F}}^{\mathrm{rCPFE}}$ (on Fig. 5) also achieves* strong *$\kappa$-relaxed function privacy (as defined in Def. 2) if* $\mathsf{aux} = \perp$ *and the dummy function vectors are chosen from the same distribution as the real ones.*

For the detailed description of the inner product rCPFE (or IP-rCPFE for short) we refer to Appendix C.

## 4.1 Performance and Possible Optimizations

For our IP-CPFE protocol, we prepared a proof of concept implementation using the Charm framework [AGM$^+$13]. To evaluate its performance in two scenarios, we compared its running times and communication costs with that of the state of the art secure arithmetic inner product computation method of the ABY framework [DSZ15]. For our experiments we used a commodity laptop with a 2.60GHz Intel® Core™ i7-6700HQ CPU and 4GB of RAM.

**Simulating regression model evaluation.** In the first use-case, we do not assume that the vectors have a special structure. The vectors to be multiplied can correspond to data and weight vectors of a binary regression model, in which case it is likely that the same model (weight vector) is evaluated over multiple inputs. Fig. 6a and 6d depict running times and overall communication costs respectively depending on the number of inputs to the same model. Fig. 6c and 6f show the cost of the dummy queries. In the same setting, our experiments show that without optimizations[4] IP-rCPFE reaches the running time of ABY for $\kappa \approx 6200$. For this scenario, we also propose a method (denoted as rCPFE opt) to pre-compute the dummy function queries of Step I. thus reducing both the online communication and computation costs. The key insight of this is that sending a value together with dummy values is essentially the same as hiding the value with a one time pad (OTP) and attaching the OTP key together with dummy keys. The gain comes from the fact that the OTP keys can be computed and sent beforehand, moreover it is enough to transmit the used seeds for a pseudo-random generator instead of the entire keys (see details in Appendix C.3). Security is not affected as long as $\mathsf{aux} = \perp$.

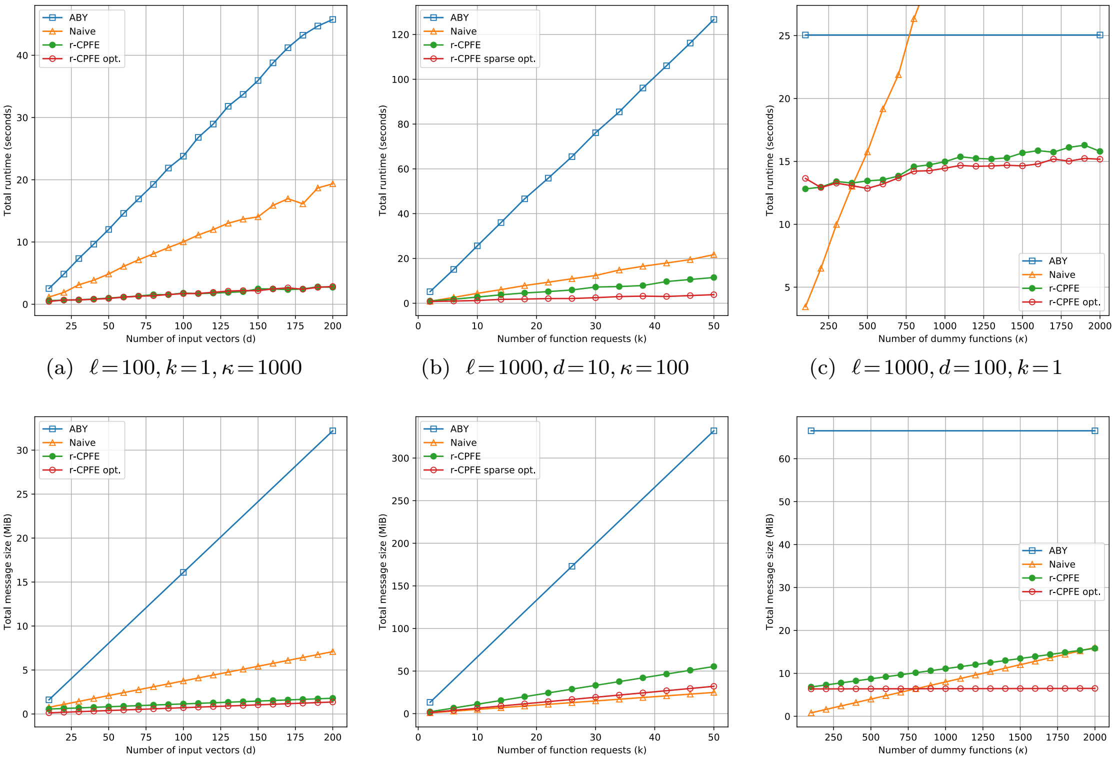
**Sparse vector products for location privacy.** The location privacy scenario of §1.2 implies the usage of sparse query vectors. Fig. 6b and 6e show how the number of queries ($k$) affects running time and message sizes respectively, when roughly 5% of the vector elements are non-zero. We note that as queries are related to real-time user requests, batching these requests, as done in Step I. of the protocol, can be unrealistic when data vectors are not changing in real time but, e.g. periodically. Because of this, in our implementation, we allowed $P_2$ to repeat Step I. for a single function and $P_1$ to answer the queries independently of encrypting the data.[5] While sparsity disables the above optimization, after masking the places of non-zero elements, the above idea can be extended for sparse vectors as long as other structural properties are not known about the vector in form of auxiliary information. For more details on the optimized variants, we refer to Appendix C.3.

# 5 Conclusion and Open Directions

In this work, we attempted to draw attention to the problem of possibly sensitive partial information leakage in the context of private function evaluation. We proposed a definitional framework for protocols that aim to prevent such leakage and showed both generic and concrete protocols to

[4] We note that while our implementation is only a proof of concept without any code level optimization, ABY has a very efficient and parallelizable implementation.

[5] It means that (3)–(4) of Step II., and Step III. are repeated until the input data changes at the end of the period.



(a) $\ell = 100, k = 1, \kappa = 1000$

(b) $\ell = 1000, d = 10, \kappa = 100$

(c) $\ell = 1000, d = 100, k = 1$

(d) $\ell = 100, k = 1, \kappa = 1000$

(e) $\ell = 1000, d = 10, \kappa = 100$

(f) $\ell = 1000, d = 100, k = 1$

Figure 6: Comparisons of the overall running times (6a–6c) and communication costs (6d–6f) of our rCPFE protocols with the ABY framework [DSZ15] and the naive OT-based approach for inner product computation ($\ell$ denotes vector dimension, $d$ and $k$ are the number of input and "function" vectors, while $\kappa$ is the number of dummy vectors).

solve the problem. The main advantage of our FE-based protocol is that it turns the privacy sacrifice required by controllability into performance improvement whenever more function evaluations are necessary.

Our work also leaves open several problems for future work. For instance, it would be important to investigate the effects of having different types of auxiliary information about the evaluated functions. Transmission and verification of dummy functions can be serious bottlenecks in our rCPFE in case of complex functions, making further efficiency improvements desirable. A first step towards this could be to find a way for restricting the set of forbidden functions – as most often very simple functions are the only undesired ones. Finally, looking for different trade-offs between function privacy and efficiency can also be interesting direction for future work.

## Acknowledgements

We would like to thank the valuable advices of Gergely Biczók and Gergely Ács on the presented application scenarios. This work was partially performed in the frames of the projects no. FIEK_16-1-2016-0007 and no. 2017-1.3.1-VKE-2017-00042, implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the FIEK_16 and the 2017-1.3. funding schemes, and it was also partially supported by the National Research, Development and Innovation Office – NKFIH, under grant contract no. 116675 (K). The first author was also supported by the Sándor Csibi Grant.

# References

[ABCP15] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In Jonathan Katz, editor, *Proceedings of Public-Key Cryptography - PKC 2015*, volume 9020 of *LNCS*, pages 733–751. Springer, 2015.

[AGM$^+$13] Joseph A. Akinyele, Christina Garman, Ian Miers, Matthew W. Pagano, Michael Rushanan, Matthew Green, and Aviel D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *J. Cryptographic Engineering*, 3(2):111–128, 2013.

[ALS16] Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 333–362. Springer, 2016.

[BGJS16] Saikrishna Badrinarayanan, Vipul Goyal, Aayush Jain, and Amit Sahai. Verifiable functional encryption. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *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 II*, volume 10032 of *Lecture Notes in Computer Science*, pages 557–587, 2016.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *Theory of Cryptography - TCC 2011. Proceedings*, volume 6597 of *LNCS*, pages 253–273. Springer, 2011.

[CIK$^+$01] Ran Canetti, Yuval Ishai, Ravi Kumar, Michael K. Reiter, Ronitt Rubinfeld, and Rebecca N. Wright. Selective private function evaluation with applications to private statistics. In Ajay D. Kshemkalyani and Nir Shavit, editors, *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing, PODC 2001,*, pages 293–304. ACM, 2001.

[CT05] Cheng-Kang Chu and Wen-Guey Tzeng. Efficient $k$-out-of-$n$ oblivious transfer schemes with adaptive and non-adaptive queries. In Serge Vaudenay, editor, *Public Key Cryptography - PKC 2005, 8th International Workshop on Theory and Practice in Public Key Cryptography, Les Diablerets, Switzerland, January 23-26, 2005, Proceedings*, volume 3386 of *Lecture Notes in Computer Science*, pages 172–183. Springer, 2005.

[DC14] Changyu Dong and Liqun Chen. A fast secure dot product protocol with application to privacy preserving association rule mining. In Vincent S. Tseng, Tu Bao Ho, Zhi-Hua Zhou, Arbee L. P. Chen, and Hung-Yu Kao, editors, *Advances in Knowledge Discovery and Data Mining - 18th Pacific-Asia Conference, PAKDD 2014, Tainan, Taiwan, May 13-16, 2014. Proceedings, Part I*, volume 8443 of *Lecture Notes in Computer Science*, pages 606–617. Springer, 2014.

[DSZ15] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015.* The Internet Society, 2015.

[FJT13] Pierre-Alain Fouque, Antoine Joux, and Mehdi Tibouchi. Injective encodings to elliptic curves. In Colin Boyd and Leonie Simpson, editors, *Information Security and Privacy - 18th Australasian Conference, ACISP 2013, Brisbane, Australia, July 1-3, 2013. Proceedings*, volume 7959 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2013.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 40–49. IEEE Computer Society, 2013.

[GGHZ16] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Functional encryption without obfuscation. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part II*, volume 9563 of *Lecture Notes in Computer Science*, pages 480–511. Springer, 2016.

[GLLM04] Bart Goethals, Sven Laur, Helger Lipmaa, and Taneli Mielikäinen. On private scalar product computation for privacy-preserving data mining. In Choonsik Park and Seongtaek Chee, editors, *Information Security and Cryptology - ICISC 2004, 7th International Conference, Seoul, Korea, December 2-3, 2004, Revised Selected Papers*, volume 3506 of *Lecture Notes in Computer Science*, pages 104–120. Springer, 2004.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 162–179. Springer, 2012.

[HBSN19] Máté Horváth, Levente Buttyán, Gábor Székely, and Dóra Neubrandt. There Is Always an Exception: Controlling Partial Information Leakage in Secure Computation. In Jae Hong Seo, editor, *Information Security and Cryptology – ICISC 2019*, LNCS, Cham, 2019. Springer International Publishing.

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 818–829. ACM, 2016.

[KKW17] W. Sean Kennedy, Vladimir Kolesnikov, and Gordon T. Wilfong. Overlaying conditional circuit clauses for secure computation. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 499–528. Springer, 2017.

[KS08] Vladimir Kolesnikov and Thomas Schneider. A practical universal circuit construction and secure evaluation of private functions. In Gene Tsudik, editor, *Financial Cryptography and Data Security, 12th International Conference, FC 2008, Cozumel, Mexico, January 28-31, 2008, Revised Selected Papers*, volume 5143 of *Lecture Notes in Computer Science*, pages 83–97. Springer, 2008.

[KS16] Ágnes Kiss and Thomas Schneider. Valiant's universal circuit is practical. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 699–728. Springer, 2016.

[NAP+14] Muhammad Naveed, Shashank Agrawal, Manoj Prabhakaran, XiaoFeng Wang, Erman Ayday, Jean-Pierre Hubaux, and Carl A. Gunter. Controlled functional encryption. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 2014*

*ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 1280–1291. ACM, 2014.

[PSS09] Annika Paus, Ahmad-Reza Sadeghi, and Thomas Schneider. Practical secure evaluation of semi-private functions. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *Applied Cryptography and Network Security, 7th International Conference, ACNS 2009, Paris-Rocquencourt, France, June 2-5, 2009. Proceedings*, volume 5536 of *Lecture Notes in Computer Science*, pages 89–106, 2009.

[RR17] Peter Rindal and Mike Rosulek. Improved private set intersection against malicious adversaries. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 235–259, 2017.

[Tze04] Wen-Guey Tzeng. Efficient 1-out-of-n oblivious transfer schemes with universally usable parameters. *IEEE Trans. Computers*, 53(2):232–240, 2004.

[Val76] Leslie G. Valiant. Universal circuits (preliminary report). In Ashok K. Chandra, Detlef Wotschke, Emily P. Friedman, and Michael A. Harrison, editors, *Proceedings of the 8th Annual ACM Symposium on Theory of Computing, May 3-5, 1976, Hershey, Pennsylvania, USA*, pages 196–203. ACM, 1976.

[ZWH+15] Youwen Zhu, Zhikuan Wang, Bilal Hassan, Yue Zhang, Jian Wang, and Cheng Qian. Fast secure scalar product protocol with (almost) optimal efficiency. In Song Guo, Xiaofei Liao, Fangming Liu, and Yanmin Zhu, editors, *Collaborative Computing: Networking, Applications, and Worksharing - 11th International Conference, CollaborateCom 2015, Wuhan, China, November 10-11, 2015. Proceedings*, volume 163 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 234–242. Springer, 2015.

# Appendix

## A Simulation Security of Functional Encryption

For completeness we recall the simulation security of FE as defined in [GVW12].

**Definition 3** ($q$-NA-SIM and $q$-AD-SIM Security of FE)**.** *Let $\mathcal{FE}$ be a functional encryption scheme for a circuit family $\mathcal{C} = \{\mathcal{C}_\nu : \mathcal{X}_\nu \to \mathcal{Y}_\nu\}_{\nu \in \mathbb{N}}$. For every PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and a PPT simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ consider the following two experiments:*

```
$\mathsf{Exp}^{\mathsf{real}}_{\mathcal{FE},\mathcal{A}}(\lambda)$
$(\mathsf{pp_{FE}}, \mathsf{msk_{FE}} \leftarrow\!\!\$\ \mathsf{FE.Setup}(\lambda)$
$(x, \mathsf{st}) \leftarrow\!\!\$\ \mathcal{A}_1^{\mathsf{FE.KeyGen}(\mathsf{msk_{FE}},\cdot)}(\mathsf{pp_{FE}})$
$\mathsf{ct} \leftarrow\!\!\$\ \mathsf{FE.Enc}(\mathsf{pp_{FE}}, x)$
$\beta \leftarrow\!\!\$\ \mathcal{A}_2^{O(\mathsf{msk_{FE}},\cdot)}(\mathsf{pp_{FE}}, \mathsf{ct}, \mathsf{st})$
$\mathsf{output}(\beta, x)$
```

```
$\mathsf{Exp}^{\mathsf{ideal}}_{\mathcal{FE},\mathcal{S}}(\lambda)$
$(\mathsf{pp_{FE}}, \mathsf{msk_{FE}}) \leftarrow\!\!\$\ \mathsf{FE.Setup}(\lambda)$
$(x, \mathsf{st}) \leftarrow\!\!\$\ \mathcal{A}_1^{\mathsf{FE.KeyGen}(\mathsf{msk_{FE}},\cdot)}(\mathsf{pp_{FE}})$
    – Let $(C_1, \dots, C_q)$ be $\mathcal{A}_1$'s oracle queries
    – Let $\mathsf{fsk}_{f_i}$ be the oracle reply to $C_i$
    – Let $\mathcal{V} := \{y_i = C_i(x), C_i, \mathsf{fsk}_{f_i}\}$.
$(\mathsf{ct}, \mathsf{st}') \leftarrow\!\!\$\ \mathcal{S}_1(\mathsf{pp_{FE}}, \mathcal{V}, \lambda)$
$\beta \leftarrow\!\!\$\ \mathcal{A}_2^{O'(\mathsf{msk_{FE}},\mathsf{st}',\cdot)}(\mathsf{pp_{FE}}, \mathsf{ct}, \mathsf{st})$
$\mathsf{output}(\beta, x)$
```

*We distinguish between two cases of the above experiment:*

1. *The* adaptive *case, where:*

   - *the oracle* $O(\mathsf{msk}_{\mathsf{FE}}, \cdot) = \mathsf{FE.KeyGen}(\mathsf{msk}_{\mathsf{FE}}, \cdot)$ *and*
   - *the oracle* $O'(\mathsf{msk}_{\mathsf{FE}}, \mathsf{st}', \cdot)$ *is the second stage of the simulator, namely* $\mathcal{S}_2^{U_x(\cdot)}(\mathsf{msk}_{\mathsf{FE}}, \mathsf{st}', \cdot)$ *where* $U_x(C) = C(x)$ *for any* $C \in \mathcal{C}_\nu$*.*

   *The simulator algorithm* $\mathcal{S}_2$ *is stateful in that after each invocation, it updates the state* $\mathsf{st}'$ *which is carried over to its next invocation. We call a simulator algorithm* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ admissible *if, on each input* $C$*,* $\mathcal{S}_2$ *just makes a single query to its oracle* $U_x(\cdot)$ *on* $C$ *itself. The functional encryption scheme* $\mathcal{FE}$ *is then said to be q-query-simulation-secure for one message against adaptive adversaries (q-AD-SIM secure for short) if there is an admissible PPT simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ *such that for every PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *that makes at most q queries, the following two distributions are computationally indistinguishable:*

$$\left\{\mathsf{Exp}^{\mathsf{real}}_{\mathcal{FE},\mathcal{A}}(\lambda)\right\}_{\nu\in\mathbb{N}} \overset{c}{\approx} \left\{\mathsf{Exp}^{\mathsf{ideal}}_{\mathcal{FE},\mathcal{S}}(\lambda)\right\}_{\nu\in\mathbb{N}}$$

2. *The* non-adaptive *case, where the oracles* $O(\mathsf{msk}_{\mathsf{FE}}, \cdot)$ *and* $O(\mathsf{msk}_{\mathsf{FE}}, \mathsf{st}, \cdot)$ *are both the "empty oracles" that return nothing: the functional encryption scheme* $\mathcal{FE}$ *is then said to be q-query-simulation-secure for one message against non-adaptive adversaries (q-NA-SIM secure, for short) if there is a PPT simulator* $\mathcal{S} = (\mathcal{S}_1, \perp)$ *such that for every PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *that makes at most q queries, the two distributions above are computationally indistinguishable.*

As shown by [GVW12, Theorem A.1.], in the non-adaptive setting (that we also use), $q$-NA-SIM security for one message is equivalent to $q$-NA-SIM security for many messages.

## B Proof of Theorem 2

We prove Theorem 2, by showing that the protocol of Fig. 5 fulfils the requirements of Definition 2 with the assumption that the underlying FE and OT are SIM secure against semi-honest adversaries.

*Proof.* As correctness directly follows from the correctness of the underlying FE and OT, we turn our attention towards the security requirements. We argue input and weak relaxed function privacy by showing that the view of both parties can be simulated (without having access to the inputs of the other party) using the simulators guaranteed by the SIM security of FE and OT.

**Corrupted $P_1$: Weak Relaxed Function Privacy.** Besides its input and output, the view of $P_1$ consists of the received OT messages and the function query $\mathcal{F}_R$. Simulation becomes trivial because of the fact that the output of $P_1$ also contains $\mathcal{F}_R$. Thus $\mathcal{S}_{P_1}((x_1, \ldots, x_d), \mathcal{F}_R)$ can return $\mathcal{F}_R$ and the output of the sender's simulator $\mathcal{S}^{\mathcal{S}}_{OT}$ guaranteed by the SIM security of OT. The simulated view is clearly indistinguishable from the real one.

**Corrupted $P_2$: Input privacy.** The following simulator $\mathcal{S}_{P_2}$ simulates the view of a corrupt $P_2$, that consists of its input $(f_1, \ldots, f_k)$, output $\{y'^*_{i,j} = f'_i(x_j)\}_{i\in[k],j\in[d]}$, the used randomness and the incoming messages. $\mathcal{S}_{P_2}$ first determines the index set $I^* = \{i \mid \exists j : y'_{i,j} \neq \perp\} \subseteq [k]$. Next, it sets up the parameters of the ideal experiment according to Def. 3. To do so, it samples $(\mathsf{msk}_{\mathsf{FE}}{}^*, \mathsf{pp}_{\mathsf{FE}}{}^*) \leftarrow_{\$} \mathsf{FE.Setup}(\lambda)$ and then for all $i \in I^*$ generates functional secret keys $\mathsf{fsk}^*_{f_i} \leftarrow_{\$} \mathsf{FE.KeyGen}(\mathsf{pp}_{\mathsf{FE}}{}^*, \mathsf{msk}_{\mathsf{FE}}{}^*, f_i)$. For the simulation of the FE ciphertexts (corresponding to unknown messages), we can use the FE simulator $\mathcal{S}_{FE}$ for many messages (implied by one-message $q$-NA-SIM security [GVW12]). Thus $\mathcal{S}_{FE}(\mathsf{pp}_{\mathsf{FE}}{}^*, \{y_{i,j} = f_i(x_j), f_i, \mathsf{fsk}^*_{f_i}\}_{i\in I^*, j\in[d]}, \lambda) = (\mathsf{ct}^*_1, \ldots, \mathsf{ct}^*_d)$ can be appended to the simulated view together with $\mathsf{pp}_{\mathsf{FE}}{}^*$. The incoming messages of Step III. are simulated using the OT simulator $\mathcal{S}^{\mathcal{R}}_{OT}$ for the receiver. Finally the output of $\mathcal{S}^{\mathcal{R}}_{OT}(\lambda, \{\mathsf{fsk}^*_{f_i}\}_{i\in I^*} \cup \{\perp_i\}_{i\in[k]\setminus I^*})$ is appended to the simulated view.

Now we show the indistinguishability of the real and simulated views. As the inputs and outputs are the same in both cases, we have to compare the randomness and the incoming messages. First notice that $\mathsf{pp_{FE}}$ and $\mathsf{pp_{FE}}^*$ are generated with different random choices. At the same time, these cannot be told apart as otherwise the choices were not random. The rest of the incoming messages depend on these parameters. Observe that $I^* = I \cap [k]$. The security of the used FE scheme guarantees that $(\mathsf{ct}_1^*, \dots, \mathsf{ct}_d^*)$ even together with functional keys $\{\mathsf{fsk}_{f_i}^*\}_{i \in I^*}$ are indistinguishable from $(\mathsf{ct}_1, \dots, \mathsf{ct}_d)$ with $\{\mathsf{fsk}_{f_i}\}_{i \in I \cap [k]}$. Finally, the security of the OT simulation guarantees that $(\mathsf{msg}_1^{\mathsf{OT}}, \dots, \mathsf{msg}_\kappa^{\mathsf{OT}})$ and $(\mathsf{msg}_1^{\mathsf{OT}^*}, \dots, \mathsf{msg}_\kappa^{\mathsf{OT}^*})$ are indistinguishable. This also implies that functional keys for the same functions (with respect to either $\mathsf{pp_{FE}}$ or $\mathsf{pp_{FE}}^*$) can be obtained both from the real and simulated OT messages. In other words, FE ciphertexts and functional keys are consistent in both cases (i.e. they allow one to obtain the same decryption outputs) due to the correctness of the FE simulation, which concludes our proof. □

# C Inner Product rCPFE and Its Optimizations

## C.1 Assumption

For completeness, we recall the classical DDH assumption on which we base the security of our rCPFE for inner product computation.

**Assumption 1** (DDH)**.** *Let $\mathbb{G}$ be a multiplicative group of prime order $p$, $g \in \mathbb{G}$ its generator element and $x, y \in \mathbb{Z}_p^*$ uniformly random values. We say that the Decisional Diffie–Hellman assumption holds in $\mathbb{G}$ if given $(g, g^x, g^y, g^r)$, no probabilistic polynomial time (PPT) algorithm can decide, with higher than $\frac{1}{2} + \mathsf{negl}(p)$ probability, whether $r = xy$ or $r$ is also a uniformly random value from $\mathbb{Z}_p^*$.*

## C.2 Instantiating Our Generic rCPFE for Inner Products

The instantiation of Protocol $\Pi_{\mathcal{F}}^{\mathrm{rCPFE}}$ (Fig. 5) with the DDH-based inner product FE scheme of [ALS16] and the 1 out of $n$ OT protocol of [Tze04] leads us to an inner product rCPFE protocol for $k = 1$. See details in Fig. 7. For simplicity, in our description we use the following notation: $g^{\vec{x}} = (g^{x_1}, \dots, g^{x_\ell})$ for $g \in \mathbb{G}$ and $\vec{x} \in \mathbb{Z}_p^\ell$. $\mathcal{IE}$ denotes an efficient injective encoding algorithm mapping messages $m \in \{0,1\}^\lambda$ to elements of $\mathbb{G}$, so that $m$ can be efficiently recovered from $\mathcal{IE}(m)$. For more details on injective encodings, see [FJT13].

We note that in DDH-based inner product FE schemes only a polynomial sized range of the possible inner product results can be efficiently decrypted and our protocol inherits this property.

## C.3 Optimizations of the Inner Product rCPFE

Figures 8 and 9 formally describes the optimisation ideas we sketched in §4.1.

Protocol $\Pi^{\mathrm{rCPFE}}_{\langle\cdot,\cdot\rangle}$

PARAMETERS: $\kappa, \lambda$ parametrizing security and function class $\langle\cdot,\cdot\rangle : \mathbb{Z}_p^\ell \times \mathbb{Z}_p^\ell \to \mathbb{Z}_p$

INPUTS:

- $P_1$: $\vec{x}_i = (x_{i1}, \ldots, x_{i\ell}) \in \mathbb{Z}_p^\ell$ for $i \in [d]$, $\mathcal{F}_A \subset \mathbb{Z}_p^\ell$
- $P_2$: $\vec{y} \in \mathbb{Z}_p^\ell$

OUTPUTS:

- $P_1$: $\mathcal{F}_R$ s.t. $\vec{y} \in \mathcal{F}_R \subset \mathbb{Z}_p^\ell$ and $|\mathcal{F}_R| = \kappa$
- $P_2$: $\langle x_i, y\rangle \in \mathbb{Z}_p$ if $x \notin \mathcal{F}_A$ and $\perp$ otherwise

PROTOCOL:

ONLINE PHASE

**Step I.** To initiate an inner product computation with $\vec{y}$, $P_2$ does the following:

(1) samples a random matrix $\mathbf{Y}_{\mathrm{top}} \leftarrow_{\$} \mathbb{Z}_p^{(\kappa-1)\times\ell}$ and appends $\vec{y}$ after the last row forming $\mathbf{Y} = (y_{i,j}) \in \mathbb{Z}_p^{\kappa\times\ell}$,

(2) picks a random permutation $\sigma$ on $\kappa$ elements to permute the rows of $\mathbf{Y}$ s.t. $\hat{\mathbf{Y}} = (\hat{y}_{i,j}) = (y_{\sigma^{-1}(i)j}) \in \mathbb{Z}_p^{\kappa\times\ell}$,

(3) finally, sends $\mathcal{F}_R = \hat{\mathbf{Y}}$ to $P_1$.

**Step II.** Upon receiving a function request $\mathcal{F}_R$, $P_1$

(1) chooses a group $\mathbb{G}$ of order a $\lambda$-bit prime $p$, with generators $g, h_0, h_1 \in \mathbb{G}$, selects an injective encoding $\mathcal{IE} : \mathbb{Z}_p^2 \to \mathbb{G}$, and set $\mathsf{pp} = (\mathbb{G}, p, g, h_0, h_1, \mathcal{IE}^{-1})$

(2) samples random $S \leftarrow_{\$} \mathbb{Z}_p$ and $\vec{s}, \vec{t} \leftarrow_{\$} \mathbb{Z}_p^\ell$ to form $\mathsf{msk}_{\mathsf{FE}} = (S, \vec{s}, \vec{t})$,

(3) $\forall j \in [d]$ samples random $r_j \leftarrow_{\$} \mathbb{Z}_p$ and to encrypt $\vec{x}_j$, computes

$$\mathsf{ct}_j = \left(c_j = g^{r_j}, d_j = g^{Sr_j}, \vec{e}_j = g^{\vec{x}_j + r(\vec{s}+S\vec{t})}\right)$$

(4) determines the index set of allowed vector queries $I := \{i \mid \hat{\vec{y}_i} \notin \mathcal{F}_A\}$,

(5) generate functional keys for all $i \in I : \mathsf{fsk}_{\hat{\vec{y}_i}} = (\hat{S}_i = \langle\hat{\vec{y}_i}, \vec{s}\rangle, \hat{T}_i = \langle\hat{\vec{y}_i}, \vec{t}\rangle)$,

(6) sends $\mathsf{pp}$ and $\{\mathsf{ct}_j\}_{j\in[d]}$ to $P_2$

**Step III.** $P_1$ and $P_2$ executes the following 1 out of $\kappa$ OT protocol:

(1) $P_2$ samples random $r' \leftarrow_{\$} \mathbb{Z}_p$, computes $R' = h_0^{r'} h_1^{\sigma(\kappa)}$ and sends $R'$ to $P_1$.

(2) for $i \in [\kappa], P_1$ samples $k_i \leftarrow_{\$} \mathbb{Z}_p$, prepares $m_i$ s.t. $m_i = \mathsf{fsk}_{\hat{\vec{y}_i}}$ if $i \in I$ and $m_i = \perp$ for $i \notin I$, then computes the OT messages to be sent to $P_1$:

$$\mathsf{msg}_{\mathsf{i}}^{\mathsf{OT}} = (a_i = g^{k_i}, b_i = \mathcal{IE}(m_i)\cdot(R'/h^i)^{k_i}).$$

OFFLINE PHASE

$P_2$ can evaluate the inner products by executing the following steps:

(1) to extract the functional key from the OT messages, select $\mathsf{msg}^{\mathsf{OT}}_{\sigma(\kappa)}$ and compute

$$\mathcal{IE}^{-1}\left(b_{\sigma(\kappa)}/a^{r'}_{\sigma(\kappa)}\right) = \mu$$

(2) if $\mu = \perp$ then output $\perp$, otherwise $\mu = \mathsf{fsk}_{\hat{\vec{y}}_{\sigma(\kappa)}} = \mathsf{fsk}_{\vec{y}} = (S_\kappa, T_\kappa)$,

(3) $\forall j \in [d]$ compute $(\prod_{i\in[\ell]} e_{ji}^{y_{ji}})/(c_j^{S_{\sigma(\kappa)}} \cdot d_j^{T_{\sigma(\kappa)}}) = g^{\langle\vec{x},\vec{y}_j\rangle}$,

(4) if the discrete log of $g^{\langle\vec{x},\vec{y}_j\rangle}$ is contained in a predetermined range, it is computed and returned as the output, otherwise $\perp$ is returned.

Figure 7: An instantiation of the generic rCPFE construction for the inner product functionality.

Protocol $\Pi^{\mathrm{rCPFE\ opt}}_{\langle\cdot,\cdot\rangle}$

**Parameters:** $\kappa, \lambda$ parametrizing security, function class $\langle\cdot,\cdot\rangle : \mathbb{Z}_p^\ell \times \mathbb{Z}_p^\ell \to \mathbb{Z}_p$, and a pseudorandom number generator $\mathsf{PRG} : \{0,1\}^\vartheta \to \{0,1\}^{\log_2 p}$

**Inputs:**

- $P_1$: $\vec{x}_i = (x_{i1}, \ldots, x_{i\ell}) \in \mathbb{Z}_p^\ell$ for $i \in [d]$, $\mathcal{F}_A \subset \mathbb{Z}_p^\ell$
- $P_2$: $\vec{y} \in \mathbb{Z}_p^\ell$

**Protocol:**

Precomputation

**Pre-Step I.** $P_2$ samples $q'_{i,j} \leftarrow_{\$} \{0,1\}^\vartheta$ for the PRG to compute $q_{i,j} = \mathsf{PRG}(q'_{i,j}), \forall i \in [n]$, $j \in [\ell]$. $\mathbf{Q} = (q_{i,j})$ is kept locally, while $\mathbf{Q'} = (q'_{i,j})$ is sent to $P_1$.

**Pre-Step II.** $P_1$ computes $\mathbf{Q} = (q_{i,j})$ by evaluating $\mathsf{PRG}(q'_{i,j}) = q_{i,j}, \forall i \in [n], j \in [\ell]$.

Online Phase

**Step I.** To generate function request $\mathcal{F}_R$ containing $\vec{y}$, $P_2$

(1) chooses a random index $i^* \leftarrow_{\$} [n]$,

(2) encrypts $\vec{y}$ with the $i^*$th OTP key: $\vec{y}' = (y_1 \oplus q_{i^*,1}, \ldots, y_\ell \oplus q_{i^*,\ell})$,

(3) finally, sends $\vec{y}'$, as a succinct description of $\mathcal{F}_R$, to $P_1$.

**Step II.** Upon receiving a function request $\mathcal{F}_R$, $P_1$

(0) decrypts $\mathcal{F}_R$ with all the OTP keys: $\vec{\tilde{y}}_i = (y'_1 \oplus q_{i,1}, \ldots, y'_\ell \oplus q_{i,\ell}), \forall i \in n$

(1)-(6) executes Step II. of $\Pi^{\mathrm{rCPFE}}_{\langle\cdot,\cdot\rangle}$ (Fig. 7).

**Step III.** $P_1$ and $P_2$ executes the following 1 out of $\kappa$ OT protocol:

(1) $P_2$ samples random $r' \leftarrow_{\$} \mathbb{Z}_p$, computes $R' = h_0^{r'} h_1^{i^*}$ and sends $R'$ to $P_1$.

(2) $P_1$ acts as in Step III. of $\Pi^{\mathrm{rCPFE}}_{\langle\cdot,\cdot\rangle}$ (Fig. 7).

Offline Phase

It is the same as in $\Pi^{\mathrm{rCPFE}}_{\langle\cdot,\cdot\rangle}$ (Fig. 7).

Figure 8: "OTP" optimization for the inner product rCPFE in case of uniformly distributed function vectors of limited size.

Protocol $\Pi^{\text{rCPFE sparse opt}}_{\langle\cdot,\cdot\rangle}$

**Parameters:** $\kappa, \lambda$ parametrizing security, function class $\langle\cdot,\cdot\rangle : \mathbb{Z}_p^\ell \times \mathbb{Z}_p^\ell \to \mathbb{Z}_p$, and a pseudorandom number generator $\mathsf{PRG} : \{0,1\}^\vartheta \to \{0,1\}^{\log_2 p}$

**Inputs:**

- $P_1$: $\vec{x}_i = (x_{i1}, \ldots, x_{i\ell}) \in \mathbb{Z}_p^\ell$ for $i \in [d]$, $\mathcal{F}_A \subset \mathbb{Z}_p^\ell$
- $P_2$: $\vec{y} \in \mathbb{Z}_p^\ell$

**Protocol:**

Precomputation

**Pre-Step I.** $P_2$ does the following:

(1) samples $\mathsf{PRG}$ seed $q'_{i,j} \leftarrow_{\$} \{0,1\}^\vartheta$ to compute $q_{i,j} = \mathsf{PRG}(q'_{i,j}), \forall i \in [n], j \in [\ell]$,

(2) selects $n$ random permutations $\Sigma = (\sigma_i, \ldots, \sigma_n)$, each on $\ell$ elements,

(3) $\mathbf{Q} = (q_{i,j})$ is kept locally, while $\Sigma$ and $\mathbf{Q}' = (q'_{i,j})$ are sent to $P_1$.

**Pre-Step II.** $P_1$ computes $\mathbf{Q} = (q_{i,j})$ by evaluating $\mathsf{PRG}(q'_{i,j}) = q_{i,j}, \forall i \in [n], j \in [\ell]$.

Online Phase

**Step I.** To generate request $\mathcal{F}_R$ for sparse $\vec{y}$ (containing $\delta \ll \ell$ non-zero values), $P_2$

(1) prepares the list of non-zero vector positions $(j_1, \ldots, j_\delta)$ s.t. $y_{j_m} \neq 0$, $\forall m \in [\delta]$,

(2) chooses a random index $i^* \leftarrow_{\$} [n]$,

(3) computes $Y := \{(a_m = \sigma_{i^*}(j_m), b_m = y_{j_m} \oplus q_{i^*,\sigma_{i^*}(j_m)})\}_{m \in [\delta]}$, where $\sigma_{i^*} \in \Sigma$,

(4) finally, sends $Y$, as a succinct description of $\mathcal{F}_R$, to $P_1$.

**Step II.** Upon receiving a function request $\mathcal{F}_R$, $P_1$

(0) prepares $\hat{\vec{y}}_i$ vectors by setting $\hat{y}_{i,\sigma_i^{-1}(a_m)} = b_m \oplus q_{i,a_m}$, $\forall\ m \in [\delta]$ and $\hat{y}_{i,j} = 0$ otherwise,

(1)-(6) executes Step II. of $\Pi^{\text{rCPFE}}_{\langle\cdot,\cdot\rangle}$ (Fig. 7).

**Step III.** $P_1$ and $P_2$ executes the following 1 out of $\kappa$ OT protocol:

(1) $P_2$ samples random $r' \leftarrow_{\$} \mathbb{Z}_p$, computes $R' = h_0^{r'} h_1^{i^*}$ and sends $R'$ to $P_1$.

(2) $P_1$ acts as in Step III. of $\Pi^{\text{rCPFE}}_{\langle\cdot,\cdot\rangle}$ (Fig. 7).

Offline Phase

It is the same as in $\Pi^{\text{rCPFE}}_{\langle\cdot,\cdot\rangle}$ (Fig. 7).

Figure 9: "OTP" optimization for rCPFE for the inner product functionality in case of sparse function vectors with uniformly distributed nonzero values.