On the security of data markets: controlled Private Function Evaluation

István Vajda
TU Budapest, Hungary
Email: vajda@hit.bme.hu

Abstract: The income of companies working on data markets steadily grows year by year. Private function evaluation (PFE) is a valuable tool in solving corresponding security problems. The task of Controlled Private Function Evaluation (CPFE) and its relaxed version (rCPFE) was proposed in [12]. We define an ideal functionality for the latter task and present a UC-secure realization of the functionality against static malicious parties. The core primitive is functional encryption (FE) and essentially this determines the conditions of realizability. Accordingly, in the case of adaptive FE-setting secure realization requires the accessibility of random oracle.

Keywords: private function evaluation, functional encryption, UC-security

1. Introduction

The growing importance of data is beyond question today [8], [18]. To the current technological trends (proliferation of smart devices and the internet of things (IoT)) an increasing amount of data is waiting for utilization. According to the business model of a data market, a data broker buys data from the owners and sells the collected data (possibly in processed form) to a third party that provides value-added service to its users. Serious security concerns may arise in connection with the operation of this model. We briefly mention a few real-life examples.

DNA database, contain information about the purpose of each gene. Such databases are extremely valuable and thus those are not sold on a whole, but rather clients are charged per access to the database. On the other hand, the particular DNA sequences accessed by a client (e.g. a pharmaceutical company) reveal a lot of information about the interests of the client, e.g., for which disease it is developing a medicine. Similarly, requests sent to the stock quotes database can reveal information about the investment strategy of the requester or patent search patterns can reveal sensitive business information. The database owner wants to control which subscriptions allow access to which types of information (it is quite likely that subscription prices vary with the type of accessed information). However, this must be reconciled in some way with the client's need for privacy.

Private function evaluation (PFE) is a valuable tool in solving this problem. Private function evaluation (PFE) is a two-party computation, where the input of client C is an efficiently evaluable function f, $f \in F$ (in some representation), and the input x of server S is a value from the domain of f. Client receives output f(x). The corresponding functionality defined by its I/O behaviour is the following: $(x, f) \rightarrow (-, f(x))$.

A refined related task is the Controlled PFE (CPFE) where server S as part of its input gives also a set of impermissible functions F_A, $F_A \subset F$, i.e. $((x, F_A), f) \rightarrow (-, f(x))$, if $f \notin F_A$, else abort. Note if we allowed the identity function as an input value of the client it would completely reveal the input of the server. A relaxed version of CPFE (rCPFE) is the task where client C shares his set of functions F_B with server S, i.e. $((x, F_A), (f, F_B)) \rightarrow (-, f(x))$, if $f \in F_B \setminus F_A$, else abort, [12]. Note the server can control the type of information accessible by the client by choosing the set of impermissible functions F_A appropriately. Similarly, the client reveals only a set of functions (F_B, $F_B \subset F$), from which it wants to select its input without revealing the actually selected function (f).

We study the rCPFE task in this paper. We design and analyze a protocol for this task that is Universally Composable (UC)-secure against static corruption adversary, where both parties can potentially be dishonest.

In a nutshell, the principle of operation of the protocol is as follows (told for a single request). A client has access to encrypted database records ($E(x_1),\ldots, E(x_n)$) encrypted by the server. Using an oblivious transfer sub-protocol client obtains a secret function key (sk_f) matching its input (f, $f \in F_B \setminus F_A$). Finally, client computes its output $f(x_1),\ldots,f(x_n)$ by decrypting the ciphertexts. The protocol builds on Functional Encryption (FE)-primitive as well as on commitment, oblivious transfer, and NIZK sub-protocols. The protocol requires a complex setup assumption.

The main difficulty follows from the realization of the underlying FE subtask, which in itself is an area of intense research in cryptography e.g. [3],[10],[16]. Based on these results, in the case of non-adaptive FE we can work in the standard setting, however, in the case of an adaptive input selection, the best we can hope for is a realization (of the FE primitive) in a random oracle setting [3]. On the other hand, CRS setups for the sub-protocols (commitment, OT, NIZK) of our protocol can be implemented with practical efficiency.

As the communication media for the protocol will probably be a publicly accessible network (e.g. Internet), instances of other, potentially hostile protocols (i.e. protocols designed to attack our protocol) may also use the same communication space simultaneously. Our goal is to meet the security requirements in such an execution environment, called a general concurrent environment. The other goal was to be able to design and analyse in a modular way, called modular composition. This latter goal implies the simulation-based definition of security. For all these purposes, we choose the Universal Composability (UC) approach of Canetti [4], [5].

With this paper, we wanted to contribute to the development process of security technologies on a major new field of applications the data markets.

2. Related works

Functional Encryption (FE) [4] is the basic primitive in our rCPFE multi-party computation task. In functional encryption (FE) a sender (input provider), Alice, encrypts plaintexts that a receiver, Bob (decryptor), can obtain functional evaluations of, while Charlie (setup + key generator) is responsible for initializing the encryption keys and issuing the decryption keys (secret function keys). Dishonest Bob attacks the confidentiality of Alice's messages. In our rCPFE multiparty computation task, we merge parties, Alice and Charlie, into a single party, server S.

We consider simulation-based security definitions for FE. Definitions distinguish between adaptive and non-adaptive cases. In the non-adaptive case an adversary makes all its secret key queries before receiving the challenge ciphertext whereas in the adaptive case, there is no such restriction.

Firstly, we refer to the original work [3], in particular to their adaptive, single message SIM FE definition as well as their „brute force" construction that is realizable in a programmable (simulatable) random oracle model for a large class of functions. By their impossibility result, it is impossible to achieve adaptive security (even in a non-standard setting) if the adversary can obtain an unbounded number of ciphertexts. Recall, the basic reason for that is that adaptivity requires non-committing encryption, and as a result [17] on non-committing encryption realization of adaptively secure SIM FE is impossible in the case of unbounded messages. It was also shown in [3] that non-adaptivity is a necessary condition for the realizability in standard setting.

These results were extended by several follow-up papers. Work [10] was the first to show an efficient construction in the standard model. They considered a single message and imposed an (a prori) upper bound q on the number of secret key queries. (Recall constructions in case of an unbounded number of secret-key queries are known only for very limited classes of circuit families, the most general being inner product predicates.)

They discussed both the non-adaptive (q, one)-NA-SIM and the adaptive (q, one)-AD-SIM definitions of security for functional encryption. They also showed that (q, one)-NA-SIM implies (q, many)-NA-SIM, i.e. the many messages non-adaptive simulation definition (see Th. A.1. in [10]). They showed construction in the standard model for (q, one)-NA-SIM secure FE encryption for any circuit family in NC1 (see Th. 5.1. in [10]). Consequently, this construction is also (q, many)-NA-SIM –secure.

Maurer [15] introduced a stronger (than SIM) security definition, called Composable Functional Encryption (CFE)-security (mentioned by them also as „fully adaptive security"). They extended adaptivity to choosing also the messages dependently on ciphertexts for previous messages (in contrast, the term „many messages" in [10] means a batch of messages without adaptive selection).

The adversarial model in all the above papers ([3],[10],[16]) assumed (only) a dishonest client attacking privacy. In contrast, the attack model in paper [2] includes also attacks that can be mounted by Alice, Charlie, or collusion of the two against Bob. They considered the task of Universally Composable FE (UC-secure FE). They proved that CFE-security is sufficient and necessary for UC-security in the adversarial model when decryptor (Bob) is corrupted.

Our design naturally builts on FE primitive. We have to take into account the assumptions and restrictions reviewed above concerning the security of the FE primitive. However, this must be done by taking into account our application model.

Firstly, we may get a meaningful practical database application even if we assume a priori fixed bounds on the number of secret key requests and the number of messages. Furthermore, in our case, there is not necessary to consider an adaptivity of the selection of messages as the messages correspond to records in a database, and in the base version the entire encrypted database is accessible for a client. On the other hand, we aim for an adversarial model that is richer than just assuming a semi-honest client.

The best we can hope for in the case of adaptive FE primitive is a secure realization in a random oracle setting. However, assuming an independent random oracle instance per protocol instance is even intuitively highly non-practical as a large number of different secure hash functions (believed as a practical (ad-hoc) implementation of pseudorandom function) does not exist. Therefore, we consider also a setting where different protocol instances have shared access to a global oracle functionality. In this regard, we consider global RO models, in particular programmable random GUC-oracle proposed in [7], to extend (base) UC-security to GUC-security.

A database security task related to our goal is oblivious transfer with access control (AC-OT) in [6]. Their protocol provides the following security guarantees: "only authorized users can access the record; the database provider does not learn which record the user accesses;

the database provider does not learn which attributes or roles the user has when she accesses the database". Similarity is that security of an "access-control" in our protocol also relies on appropriate security of an OT sub-protocol. However, we allow the client to learn (any permitted) efficiently computable mapping of database records (represented as binary strings). The protocol in [6] is secure by stand-alone simulation-based analysis, while we aim for UC security. However, the protocol in [6] can be realized in the standard setting.

Zk-SNARK (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge) has recently found various applications in verifiable computation and blockchain applications. Importantly, zk-SNARK exists for any NP-language. Our protocol extensively relies on NIZK proofs and we consider zk-SNARKs as efficient implementations for them. Paper [13] proposed efficient SNARK-lifting transformations that allow transforming Zk-SNARKs to zero-knowledge proofs with simulation sound extractability, such that they could be adopted in UC-secure protocols. The property of simulation sound extractability means that whenever the adversary submits a proof for a new statement, the simulator will be able to extract a valid witness except with negligible failure probability.

Report [12] is (naturally) the closest to our work. The MPC tasks considered here were introduced in [12]. Their definition of the tasks (their functionality) was as given in the introduction above. Work [12] provides preliminary thoughts about the realization of this task, including a sketchy analysis in the stand-alone setting against a static semi-honest adversary.

Contributions:
Our paper

- defines an ideal functionality (F_{rCPFE}) for the task of relaxed Controlled Private Function Evaluation (rCPFE),
- presents protocol versions for a UC-secure realization of the functionality against static malicious parties, for non-adaptive and adaptive FE settings,
- applies compositional approach extensively in the design and analysis, where a „thin" main program calls functional encryption (FE), oblivious transfer, commitment, and NIZK component algorithms,
- provides a claim on the existence of global UC secure FE in a programmable random GUC-oracle setting.

3. Assumptions

We assume that all parties are running efficient (i.e. PPT) algorithms. Clients communicate with the server over secret communication channels. Each time a client initiates a new interaction with the server (sends a new request), it does so using a fresh instance of a secret channel.

In general, we cannot realize the aimed functionality (F_{rCPFE}) in the standard cryptographic setting. In special scenarios, where non-adaptivity can be maintained with respect to FE functionality, we can work in standard cryptographic setting (by an extended meaning of this setting, i.e. without random oracle).

Adversarial model: We consider static malicious corruption adversary. The assumed goals of a malicious server include cheating with function key in such a way that the function key provided by the server does not match the request (the function received from the client), furthermore potentially dishonest computation of encryptions of database records as well as attempts to learn the private input (function f) of the client.

However, we assume that the integrity of the database content (formally the input $(x_1,\ldots,x_n)$ of the server) is guaranteed, in particular it does not vary by a malicious action during the lifetime of the database. Note, allowing a real malicious adversary that can rewrite the stored database records arbitrarily, would fundamentally destroy the security of the system, as it could set clients' output (i.e. the output of the protocol) arbitrarily in a straightforward way. The technical solution is that the server commits to its input at the beginning of the instance's run (in a model where a multi-session copy of the protocol is assigned to a database content).

A corrupt client only performs a passive attack, i.e. attacks against the privacy of the server (note, such assumption corresponds to the standard assumption for the FE task [3],[10],[16]). A client might wish to learn private database records, secret function keys.

Intractability and setup assumptions: Our protocol uses several different primitives, such as functional encryption, commitment, oblivious transfer, and NIZK. As we aim for UC-secure realization, which need UC-secure realizations. Accordingly, the intractability and setup assumptions are implicit in the sense that those are determined by the concrete realization of the primitives. More specifically, general feasibility statements formulate necessary/sufficient conditions for corresponding trusted setups. These issues will be detailed later (see in Section 5).

4. The non-adaptive case

First, we consider a base (single-session) scenario, when a database serves only a single request with a single input function f. The order of message sendings (i.e. of the ciphertexts and secret function keys) will guarantee a Non-Adaptivity (NA) setting for the FE-primitive. Specifically, client has to send its request (function f) before it sees encrypted database records. As a next step, we generalize the case of a single input function f to a single input of a batch of functions $(f_1,\ldots,f_q)$. This (straightforward) generalization can equivalently be seen as a parallelized version of the base case. A further possibility for a natural extension within the NA-setting is when different clients are served synchronously (in parallel) by the database, where the requests from different clients are collected into a single request message.

4.1 The single-session ideal functionality F_{rCPFE}

The functionality is shown in Figure 1.

Parties: S (server), C (client). Parameters: F (base set of functions)

1. Upon receiving an input (**KeyGen**, sid) from some party S, ideal functionality verifies that sid = (S, sid') for some fresh value sid'. If not so, then ignore the input. Else, it hands (**KeyGen**,sid) to the (ideal system) adversary (simulator, Sim). Sim generates public key PK. Upon receiving (**Pub_key**, *sid,* PK) from Sim, it outputs (**Pub_key**, *sid,* pk) to S.

2. Upon receiving an input message (***InicClient***, sid, F_B) from party C, it verifies that {sid=(S, sid')}. If not so, then it ignores the input. Else, it verifies that {$F_B \subseteq F$}. If so, then it stores (sid, F_B) and sends a private delayed message (***InicClient***, sid, F_B) to party *S*, else it sends message Abort1 to party S and aborts the session.

3. Upon receiving an input message (***InicServer***, sid, F_A) from party S, it verifies that {sid=(S, sid')}. If not so, then it ignores the input. Else, it verifies that {$F_A \subseteq F$}. If so, then it stores (sid, F_A) and sends a private delayed message (***InicServer***, sid, F_A) to party *S*, else it sends message Abort2 to party C and aborts the session.

4. Upon receiving an input message (***InputClient***, sid, f) from party *C*, it verifies that {sid=(S, sid'). If not so, it ignores the input, else stores (sid, f).

5. Upon the condition that {"Inic" inputs have been set for both parties} AND {an input function f from party C has already been stored} it verifies that {$f \in F_B \backslash F_A$}. If not so, then it sends message Abort3 to (honest) party S and aborts the session.

6. Upon receiving an input message (***InputServer***, sid, $x_1,\dots,x_n$) from party *S*, it verifies if {sid=(S, sid')}. If not so, then it ignores the input, else it stores (sid, $x_1,\dots,x_n$).

7. Upon the condition that {all inputs with id sid have been set for both parties}, it sends output (***Output***, sid, $f(x_1),\dots,f(x_n)$) to party C and halts the session.

8. Upon receiving ***(Corrupt,*** sid*,* ssid, C) from the adversary followed by a pair (f', F'_B) and no output has been yet written to C in session sid, then it stores (sid, f, F_B), where f=f', $F_B = F'_B$).

Upon receiving ***(Corrupt,*** sid, S*)* from the adversary and an input (sid, $x_1,\dots,x_n$, F_A) has already been received from S, ideal functionality discloses this input to the adversary. If the adversary tries to change honest output ($y_1=f(x_1),\dots, y_n = f(x_n)$) to party C to a different (well-formatted) sequence ($y'_1,\dots,y'_n$), the ideal functionality sends output Abort 4 to honest party C and terminates. (Here, a well-formatted sequence means a sequence ($y'_1=g(x'_1),\dots, y'_n = g(x'_n)$), where g, $g \neq f$ is an allowed function.)

Figure 1: Single-session ideal functionality F_{rCPFE}

Intuitively, session id sid=(S, sid') is associated with a FE key pair (PK, MK) generated by server S. A single client sends a single database request to the database.

The integrity of the database (i.e. of input $(x_1,\ldots,x_n)$) is guaranteed (it is intact over the lifetime of the database). However, a corrupted server may deviate from the specification when it processes this input via computation of encryption and function keys.

Recall, in our adversarial model we assumed a semi-honest corrupted party C. This means that such a party will output $(f(x_1),\ldots,f(x_n))$ for an input f.

In the above definition input $(x_1,\ldots,x_n)$ means the entire database content. This can be specified (and the definition of the ideal functionality accordingly modified) so that this input contains the segment of the database that the client is authorized to access (e.g. the client has subscribed to the related service).

4.2 Realization of the single-session functionality

Protocol π builds on a q-NA-SIM-secure FE primitive, furthermore on a standard secure perfect binding non-interactive commitment, an UC-secure 1-out-of-m oblivious transfer (OT), and a UC-secure Non-Interactive Zero-Knowledge (NIZK) sub-protocol.

By adapting the ideas of the classic GMW protocol compiler, the protocol starts with preliminary steps consisting of the key setup and the commitments to the inputs and random tapes. We briefly recapitulate these steps.

First, each party commits to its input. Next, parties run a special coin-tossing protocol, where one party receives a uniformly distributed string (to be used as its random tape) and the other party receives a commitment to that string (they perform these steps in both roles).

The point is that these commitments to the random tape and to the inputs are used to enforce a semi-honest adversarial behaviour. The explanation is briefly as follows. A protocol specification is a deterministic function of a party's view consisting of its input, random tape and messages received so far, furthermore that each party holds a commitment to the input and random tape of the other party. Note also that the messages sent so far are public. Therefore, an assertion that a new message is generated according to the protocol specification is an NP statement, where the party sending the message knows an adequate NP-witness to it. This implies that parties can use zero-knowledge proofs to show that their steps are indeed according to the protocol specification.

Accordingly, the definition of the protocol is as follows:

FE key setup:

Party S invokes the key generation algorithm of the FE primitive with input 1^n, where n is a security parameter: KeyGen$(1^n) \rightarrow$ (PK, MK). Party S publishes public key PK.

Commitments to inputs and random tapes:

Parties (C and S) generate their local random elements and commit to these elements as well as they commit to their inputs.

Here we can apply the technique of universally composable protocol compilation [5], an extension to the classic GMW compiler. Recall, a (straighforward) composition strategy using universally composable commitments would not work because the receiver of a universally composable commitment receives no information (except a formal 'receipt') about the committed value (recall the definition of F_{COM}). This fact implies the problem, that no NP-statement can be formulated that a party can prove relative to its input commitment.

The core of the technique in [5] is the commit-and-prove ideal functionality, F_{CP}. The "commit" phase of the F_{CP} functionality is used in order to execute the input and coin-tossing phases of the compiler. The "prove" phase of the F_{CP} functionality is used to force the adversary to send messages according to the protocol specification and consistent with the input and the random-tape resulting from the coin-tossing (as fixed in the commit phase of F_{CP}).

We use a modified version of functionality F_{CP} [5], denoting it F'_{CP}. F'_{CP} is essentially identical to F_{CP}, except the 'prove' part, where additionally the receiver is allowed to verify a single claim and also a claim chosen from a list of claims ('claim_list' in Fig 2) where the latter is done in an oblivious manner. Definition of ideal functionality F'_{CP} is shown in Fig. 2. (Parties in this functionality, committer (C) and verifier (V) correspond to server (S) and client (C) in our protocol, resp.)

Commit and Prove ideal functionality (F'_{cp}):

parameterized by a relation R, parties committer C, receiver V and adversary A:

1. *Upon receiving (sid, C, V, "**commit**", w) from (sid, C), add w to the list W of committed values, and output (sid, C, V, "receipt") to (sid, V) and A. (Initially, list W is empty.)*

2. *Upon receiving (sid, C, V, "**claim_list**", $Y=(y_0, y_1,...,y_m)$), $m\geq 1$ from (sid, C), store (sid, Y).*

3. *Upon receiving (sid, C, V, "**prove**", y, i), $i\geq 0$ from (sid, V) then do as follows:*
if i=0 then send (sid, C, V, y_0, $R(y_0, W)$) to A and to (sid, V), otherwise
if (sid, claim_list) is not stored or index i is not within range $1\leq i \leq |claim\_list|$ then ignore input, else send send (sid, C, V, y_0, y_i, $R(y_0, W) \cap R(y_i, W)$) to A and to (sid, V).

Fig. 2: Ideal functionality F'_{CP}

The 'prove' instruction has a parameter i. If i=0 then this instruction corresponds to a proof of a (single) claim y_0. The 'prove' instruction with parameter value i>0 is a combined case of a single claim and a claim_list, where from the latter a claim is chosen in oblivious manner. This parameter setting is used to prove the honest computation of encryptions (claim y_0) and secret function keys (claims $y_1,\dots,y_m$).

We assume that ideal functionality F'_{cp} involves the commitment to the random element generation and the inputs, both. Here we do not go into more details as it is precisely described in [5].

The F'_{CP}-hybrid protocol π is defined in Figure 3. Recall, messages are assumed to be sent over secret channel.

Preliminary steps:

01. Party S calls KeyGen(1^n) → (PK, MK) algorithm of the FE scheme and publishes PK.

02. Parties execute the "commit" phase of ideal functionality F'_{CP}

The main part of the protocol:

1. C → S: F_B	: party S verifies if $F_B \subseteq F$ and if not, then S outputs Abort1 and aborts
2. S → C: F_A	: party C verifies if $F_A \subseteq F$ and if not, then C outputs Abort2 and aborts
3. C → F'_{CP}	: (sid, S, C, "*prove*", claim-list elements with index 0 and index i)
F'_{CP}	: if i ∉ {1,...,m} then F'_{CP} sends an abort message to (honest) S, else it proceeds. If S receives abort message, it sends output Abort 3 and aborts.
S	: party S calls and the encryption and the function key generator algorithms of the FE scheme and computes ciphertexts $E(x_1),\ldots,E(x_n)$ as well as function keys $sk_{f1}, \ldots , sk_{fm}$, where$\{f_1,\ldots,f_m\}= F_B \backslash F_A$
4. S → F'_{CP}	: claim-list $Y=(y_0,y_1,\ldots,y_n)$, where y_0= ENC=[$E(x_1),\ldots,E(x_n)$, claim2], where claim2 = {encryptions $E(x_1),\ldots,E(x_n)$ 'were computed honestly' by party S }; $(y_1,\ldots,y_n)$= FK=[(sk_{f1}, claim1$_{f1}$), … , (sk_{fm}, claim1$_{fm}$)], $\{f_1,\ldots,f_m\}= F_B \backslash F_A$,where claim1$_{fi}$={function key sk_{fi} , $f_i \in F_B \backslash F_A$ 'was computed honestly' by party S}, i=1,…,m.
5. F'_{CP} → C	: (upon all input received) (claim2, claim1$_{fi}$, b), b∈{0,1}
C	: if b=0, then C outputs message Abort4 and aborts, otherwise it outputs $D_{skfi}(E(x_1)),\ldots, D_{skf}(E(x_n))$

Figure 3: The F'_{CP}-hybrid protocol π for realization of single-session functionality F_{rCPFE}

The role of proofs is to enforce the corrupt server to generate function keys and ciphertexts in an honest way. Party S proves that when party C uses a function key sk_f an encrypted record E(z), ∀z∈X (input space X) then it decodes into f(z). Similarly, party S proves that the series of encrypted records $E(x_1),\ldots,E(x_n)$ are honestly computed mappings of inputs $x_1,\ldots,x_n$ and r, where S is committed to random value r in the preliminary phase.

Our claim about the security of hybrid protocol π is as follows:

Theorem 1: The F'_{CP} - hybrid protocol π is UC-secure againts a static, malicious adversary (defined by the assumptions), assuming a q-NA-SIM-secure FE primitive.

Proof: We define a straight-line black-box simulator in stand-alone setting. Recall every protocol that is secure in stand-alone setting with a straight-line black-box simulator remains secure under concurrent general composition with non-adaptive inputs. The corresponding simulation and its analysis are detailed in Section 4.3 below. ■

Corollary to Theorem 1: Full-fetched protocol π is UC-secure against a static, malicious adversary, assuming UC-secure realization of F'_{CP} ideal subroutine (against a static, malicious adversary).

Proof: We refer to the universal composition theorem [4], [5]. ■

4.3 Analysis of the F'_{CP} -hybrid protocol

We show that for arbitrary efficient adversary there exists an efficient straight-line simulator Sim.

4.3.1 Definition of the simulator

Simulator Sim runs the FE-simulator. In particular, it can run the key generation (*KeyGen*: $1^n \rightarrow (PK, MK)$), the function key simulation (*func_KeyGen*: $f \rightarrow f_{sk^*}$, $f \in F$) and the ciphertext simulation (*Encrypt*: $x \rightarrow c^*$, $x \in X$) algorithms of the FE-simulator. Simulator Sim simulates ideal functionality F'_{CP} as well as key generation algorithm *KeyGen* for party S.

We show the steps of simulation by the steps of the protocol.

Case 1: corrupt S

Pre-processing steps:

FE key setup phase: Party S receives an input (**KeyGen**, sid) (from the calling environment). Sim forwards this input to party S. Upon receiving a call to key generation algorithm *KeyGen* from S, simulator Sim runs the algorithms and hands public key PK to party S. At the same time Sim forwards input (**KeyGen**, sid) also to functionality F_{rCPFE}, and for the corresponding call from the functionality Sim simulates the same public key PK. When F_{rCPFE} outputs public key PK to party S, simulator Sim outputs PK to the calling environment.

Commitments to inputs and local random elements:

Simulator (Sim) simulates the "commit" phase of the F'_{CP} functionality for party S. Sim learns the witnesses, i.e. the inputs and local random elements of party S.

Simulation of the main part of F'_{CP} -hybrid protocol:

If Sim receives an output message Abort1 to party S from ideal functionality F_{rCPFE} then Sim forwards it to S and halts.

Step 1. Sim(C)→S:

Party C sends an input F_B to functionality F_{rCPFE} that is output to party S by the functionality. Sim learns this output and forwards it to party S.

Step 2. S→Sim(C):

Party S sends a message F_A to party C. Sim captures the message and sends it as input to ideal functionality F_{rCPFE} on behalf of S via external interaction.

If Sim receives an output message Abort3 to party S from ideal functionality F_{rCPFE} then Sim forwards it to S and halts.

Step 4. S→Sim(F'$_{CP}$):

Party S sends input message (ENC, FK) to F'_{CP} via internal simulated interaction. Sim captures the message and stores it.

Sim verifies claims in messages ENC and FK (recall, SIM knows the corresponding witnesses). If any of these claims does not verify then Sim forces an output Abort4 from ideal functionality F_{rCPFE} for party C. It is done as follows:

Sim sends input $x=(x_1,\ldots,x_n)$ to ideal functionality F_{rCPFE} on behalf of S via external interaction. Next, Sim (ideal system's adversary) sends message ***(Corrupt,*** sid, S*)* to ideal functionality F_{rCPFE} on. When Sim receives message $(y_1=f(x_1),\ldots, y_n = f(x_n))$ from the ideal functionality Sim changes it to a different (well-formatted) sequence $(y'_1,\ldots,y'_n)$ and sends it to the ideal functionality. When the latter receives this message it sends output Abort 4 to honest party C and halts.

Case 2: corrupt C

There is a preliminary step of key generation similar to the case of corrupt S.

Step 1. C→Sim(S):

Party C sends a message F_B to party C. Sim captures the message and sends it as input to ideal functionality F_{rCPFE} on behalf of C via external interaction.

Step 2. Sim(S)→C:

Party S sends an input F_A to functionality F_{rCPFE} that the functionality outputs to party C. Sim learns this output and forwards it to party C.

Step 3. C→ Sim(F'$_{CP}$):

Party C sends input index-pair (0, i) to F'_{CP} via internal simulated interaction. If $i \notin \{1,\ldots,m\}$ then Sim sends f_i as input to ideal functionality F_{rCPFE} on behalf of C via external interaction, otherwise Sim sends arbitrary function $f^*\in F$, $f^*\notin F_B\backslash F_A$ as input to the ideal functionality.

Step 5. Sim(F'$_{CP}$)→C:

When ideal functionality F_{rCPFE} sends output message to party C then Sim does as follows:
if the output message is $(f_i(x_1),\ldots,f_i(x_n))$ then SIM sends message (claim2, claim1$_{f_i}$, 1) to party C, otherwise, when the output message is Abort 4 then SIM sends message (claim2, claim1$_{f_i}$, 0).

■

4.3.2 Proof of the success of simulation

By definition the view of a corrupted party (adversary) at a step of a protocol: {input, random tape, messages received so far}. We perform (stand-alone) straight-line simulation so that message transmissions will be in-sync with the order of the messages in the real system. Therefore, it is sufficient to examine the indistinguishability of the views of the adversary only at the end of the run. We distinguish the executions with and without an abort event.

Case of no abort

Message views of corrupted parties are as follows (received messages are shown in time order of their arrival):

Ideal system (F_{rCPFE}):
Party S is corrupt: F_B
Party C is corrupt: F_A, $(f_i(x_1),\ldots,f_i(x_n))$

Real system (F'_{CP} -hybrid protocol):
Party S is corrupt: F_B
Party C is corrupt: F_A, $(f_i(x_1),\ldots,f_i(x_n))$

Case of abort: We have to show the indistinguishability of the two systems in case of aborts. Concretely, we show that corresponding probabilities of aborts are equal in the two systems. Note this is obvious for abort events caused by erroneous inputs (Abort1-3).

In the real system, an abort event Abort4 happens if a claim (in ENC or FK) does not verify as correct. In the ideal system, an abort event Abort4 happens if the adversary tries to change honest output ($y_1=f(x_1),\ldots,y_n = f(x_n)$) to party C to a different (well-formatted) sequence ($y'_1=f'(x'_1),\ldots,y'_n = f'(x'_n)$). Here we observe that such change may arise by dishonest computation of encryptions or function keys.■

4.4 Parallelized version

A frequent (i.e. per database query) change of the FE key pair (PK, MK) limits the application possibilities of the single-session scenario. We can significantly reduce this disadvantage by making the client interested in sending its requests to the database at the same time in a batch (e.g. by offering a more favorable price of service per request in an application scenario). Note that with such extension, we are still in the non-adaptive FE scenario.

The formal definition of the corresponding (parallelized) version of ideal functionality F_{rCPFE} is the same as the base functionality excepts a few straightforward differences:

Input f of the client is replaced by a string $f'=(f_1,\ldots,f_q)$ of functions, where $f_i \in F$, $i=1,\ldots,q$ with a priori fixed value of q. The F'_{CP} ideal functionality processes the elements of the input sequence f' serially, i.e. one by one. Similarly, the necessary changes to the hybrid protocol (Figure 3) are straightforward, so from space-saving reasons, those we do not detail here.

The technical details of the analysis for the parallel version are essentially the same as for the base case. The claim of Theorem 1 remains valid.

5. The adaptive case

In a general application scenario, an adaptive selection of inputs by clients is unavoidable. Indeed, in a natural practical scenario, database content is accessible for a longer period for asynchronous clients, furthermore clients can turn to the server multiple times. This, in turn, implies that in general the protocol cannot be implemented in a standard setting, since the fundamental primitive of the protocol, a simulation-secure FE primitive cannot be realized in such a setting. Random oracle setup is necessary.

5.1 The multi-session ideal functionality F'_{rCPFE}

Definition of multi-session ideal functionality F'_{rCPFE} is shown in Figure 4. At first glance, this looks the same as the definition of the single-session functionality. The difference is that by the multi-session functionality F'_{rCPFE}, the database can be queried an arbitrary number of times by an arbitrary number of clients under the same main session identifier sid but under different fresh sub-session identifiers (ssid). Intuitively, the (main) session id value sid=(S, sid') corresponds to a fresh FE key pair (PK, MK) generated by server S. We assume that a new pair of such keys is generated for a new database. Sub-session identifier ssid=(C, S, ssid') corresponds to the ssid'-th instance of the functionality run by parties C and S. A natural choice is to bind ssid to a fresh instance of communication keys (i.e. to a fresh instance of secret channel) between C and S.

Rule1 of functionality F'_{rCPFE} opens the (main) session with a fresh (main) identifier sid (= (S, sid')). Rule 2 opens sub-session with a fresh sub-identifier ssid=(C, S, ssid'). In further rules verification of (sid, ssid) of messages refers to the verification that the identifier is identical to the identifier set for the actual instance in Rule 2, furthermore that no input message (of the actual type) with id (sid, ssid) has already been accepted by the instance. Furthermore, a sub-session is aborted if id (sid, ssid) of an input message is correct however its payload is invalid.

1. Upon receiving an input message (***KeyGen***, sid) from some party S, ideal functionality verifies that sid = (S, sid') for some fresh value sid'. If not so, then it ignores the input. Else, it hands (***KeyGen***,sid) to the ideal system's adversary (simulator, Sim). Sim generates public key PK. Upon receiving (***Pub_key***, sid, PK) from Sim, it outputs (***Pub_key***, *sid,* PK) to S.

2. Upon receiving an input message (***InicClient***, sid, ssid, F_B) from party *C*, it verifies that {sid=(S, sid') and that ssid=(C, S, ssid'), where ssid' is a fresh value. If not so, then it ignores the input. Else, it verifies that {$F_B \subseteq F$}. If so, then it stores (sid, ssid, F_B) and sends a private delayed message (***InicClient***, sid, F_B) to party *S*. Else, it sends message Abort1 to party S and aborts sub-session (sid, ssid).

3. Upon receiving an input message (***InicServer***, sid, ssid, F_A) from party *S*, it verifies that {sid=(S, sid')}and ssid=(C, S, ssid') such that {an *InicClient* message with sub-sid value ssid has already been received}. If not so, then it ignores the input. Else, it verifies that {$F_A \subseteq F$}. If so, then it stores (sid, ssid, F_A) and sends a private delayed message (***InicServer***, sid, F_A) to party C. Else, it sends message Abort2 to party C and aborts sub-session (sid, ssid).

4. Upon receiving an input message (***InputClient***, sid, ssid, f) from party *C*, it verifies that {sid=(S, sid')}and ssid=(C, S, ssid'). If not so, then it ignores the input. Else, it stores (sid, ssid, f).

5. Upon the condition that {*Inic* inputs with id (sid, ssid) have been set for both parties}AND {an input function f from party C has already been stored} it verifies that {$f \in F_B \backslash F_A$}. If not so, then it sends message Abort3 to party S and aborts sub-session (sid, ssid).

6. Upon receiving an input message (***InputServer***, sid, ssid, $x_1,\ldots,x_n$) from party *S*, it verifies that {sid=(S, sid')} and ssid=(C, S, ssid'). If not so, then it ignores the input, else it stores (sid, ssid, $x_1,\ldots,x_n$).

7. Upon the condition that {all inputs with id (sid, ssid) have been set for both parties} it sends output (***Output***, sid, ssid, $f(x_1),\ldots,f(x_n)$) to party C.

8. Upon receiving ***(Corrupt,*** sid, ssid, C) from the adversary followed by a pair (f', F'_B) and no output has been yet written to C in sub-session (sid, ssid), then it stores (sid, ssid, f, F_B), where f=f', $F_B = F'_B$). Ideal functionality outputs to party C whatever the adversary outputs.

Upon receiving ***(Corrupt,*** sid, ssid, S*)* from the adversary and an input (sid, ssid, $x_1,\ldots,x_n$, F_A) has already been received from S, ideal functionality discloses this input to the adversary. If the adversary tries to change honest output ($y_1=f(x_1),\ldots, y_n = f(x_n)$) to party C to a (well-formatted) sequence ($y'_1,\ldots,y'_n$), the ideal functionality sends output Abort 4 to honest party C and terminates. (Here, a well-formatted sequence means a sequence ($y'_1=g(x'_1),\ldots, y'_n = g(x'_n)$), where g, $g \neq f$ is an allowed function.)

Figure 4: Ideal functionality F'_{rCPFE}

5.2 UC-secure realization in the adaptive case

Because we want UC-security, a realization of each subtask (FE and F'$_{CP}$ (F$_{OT}$, F$_{ZK}$)) requires an appropriate trust-based setup. The NIZK and OT subprotocols can UC-securely be implemented in their respective CRS model (see also chapter 6). Additionally, each protocol instance uses its own independent random oracle instance, called the local random oracle model. (It is local to instance with id sid, i.e. it serves all client-server sessions with identifier (sid, ssid) for some ssid.) In the proof of security, the simulator simulates (also) the corresponding F$_{RO}$ functionality, which enables the simulator to learn the queries of all involved parties and to program any "random-looking" (indistinguishable from truly random) values as outputs. Let the corresponding F'$_{CP}$, F$_{RO}$ –hybrid protocol be denoted π'.

The analysis goes in F'$_{CP}$, F$_{RO}$ –hybrid. The key technical points of the simulation and its analysis are essentially the same as in the single-session case, therefore we do not detail it here due to space-saving. Our claim is as follows.

Theorem 2: (F'$_{CP}$, F$_{RO}$)-hybrid protocol π' is UC-secure against a static, malicious adversary (defined by our assumptions), assuming an AD-SIM-secure FE primitive.

Proof (Sketch): It is essentially similar to the proof of Theorem 1 with the following difference. The FE simulator needs to access a local random oracle. Simulator Sim simulates functionality F$_{RO}$ for the FE simulator. ■

Corollary to Theorem 2: Full-fetched protocol π' is UC-secure against a static, malicious adversary in random oracle model, assuming an UC-secure realization of F'$_{CP}$ ideal functionality (against a static, malicious adversary).

Proof: We refer to the universal composition theorem [4], [5]. ■

5.2.1 UC realization of the F'$_{cp}$ functionality

We summarize the steps of realization in a nutshell. First, we UC realize the ideal functionality F'$_{CP}$ (against static adversaries) as an F$_{OT}$-hybrid protocol, i.e. we assume that an 1-out-of-m ideal oblivious transfer functionality is accessible for parties. Let ρ denote the F$_{OT}$-hybrid protocol. We realize protocol ρ is three steps:

First we design protocol ρ' such that it is UC realization of ρ in F_{zk}-hybrid model, that essentially means realization of the commitment subtask. In case of a static adversary we can use a standard (perfectly binding, non-interactive) commitment scheme. Recall, the use of ZK-proof in the commit phase lifts the standard-secure commitment to the level of UC-security (in particular, it adds the capability of extraction of the committed value).

The next step of realization of protocol ρ is the realization of ideal functionality F_{zk} with a UC-secure NIZK protocol.

Finally, ideal functionality F_{OT} is realized UC-securely.

6. UC-realization of components

6.1 The FE primitive

The main difficulty with the realization of ideal functionality F'_{rCPFE} is the construction of the FE with adequate security. Assuming AD-SIM-secure FE constructions we can implement a UC-secure protocol in the non-standard setting of a TTP running random-oracle functionality F_{RO}.

Different instances of the protocol, i.e. instances with different identifier sid (see the definition of ideal functionality F_{rCPFE} for main identifier sid) have an own instance of a random oracle (called as local). By our example scenario different databases correspond to different sid values, i.e. different database have an own instance of a random oracle. The question arises whether it is possible to use a global oracle, serving different service providers or a service provider with different databases.

A proof-technical obstacle arises in this case. Recall, in the GUC framework, the (calling and distinguishing) environment also has direct access to the global oracle and this has serious consequences: the global version of F_{RO} functionality is the strictest among all RO models. The simulator is neither allowed to observe the environment's random-oracle queries nor to program its answers. Therefore, to give an advantage to the simulator seems impossible. However, it turned out (see in [7]) that even such strict functionality allows GUC-secure practical constructions for digital signatures and public-key encryption. The breakthrough observation was (see the details in [7]) that the (classic) proof technique of rewinding (of the oracle) can be applied in the part of the proof about indistinguishability, instead of within the simulation algorithm (where the rewinding tool is usually applied). Our point here is that we can apply the mentioned proof technique from [7] also in our task:

Theorem 3: *There exists global universally composable (GUC-secure) adaptively secure FE scheme in programmable random GUC-oracle setting.*

Proof (Sketch): Boneh's AD-Sim-secure brute force construction [3] is GUC-secure in programmable random GUC-oracle model defined in [7]. To prove it a proof technique from [7] (see it in Ch. 4.2 of [7]) can be adapted. ■

6.2 The NIZK subprotocol

Recall we need UC-secure NIZK and OT protocols to realize the ideal functionality F'_{cp}.

Zk-SNARKs are succinct NIZKs (i.e. have a short proof and a fast verifier). Probably the main (theoretical and practical) advantage of Zk-SNARK is that it exists for any NP-language. Accordingly, Zk-SNARK proofs exist for implementation of NIZK-proofs also in our protocol. One additional problem remains. We would like to use SNARKs in a UC-secure protocol. However, the SNARK definition is not directly compatible with this framework, due to its use of non black-box knowledge extraction. Fortunately this problem is solved, because SNARKs can be transformed into UC-secure NIZKs with tolerable overhead [14].

Zk-SNARKs rely on a common reference string (CRS). It assumes that in the setup phase of the protocol a trusted party publishes a CRS, sampled from some specialized distribution, while not leaking any side information. It is well known that subverting the setup phase can make it easy to break the security, in particular, leaking a CRS trapdoor makes it trivial to prove false statements. We return to weakened trust assumptions in chapter 7.

6.3 The OT subprotocol

In paper [9] an efficient, universally composable oblivious transfer (OT) was proposed, where a single, 'global', common reference string (CRS) can be used for multiple invocations of oblivious transfer by arbitrary pairs of parties. It is also round-optimal (3 rounds) with static security. A complexity comparison of UC-secure OT constructions can also be found in [9] (see Table 1. p.5).

7. Complexity issues and practical applicability

7.1 A time complexity bottleneck and solution

The main practical disadvantage of protocol π in Figure 3 is its potentially high time complexity. This is due to the large number of (NIZK-) proofs, caused by the potentially large number of different function keys. Let's look at a specific numerical example. If using a commercial processor and it takes about 1 minute to calculate each proof, then with a setting of e.g. m=1000 possible functions in a session, it would mean in itself a processing time of the order of one day (within the runtime of a session).

One remedy for this problem (while keeping the basic principles of the protocol) is that the server computes the function keys for the total set F of function as a preprocessing step. In an example scenario, the server (with a given database) would update the keys on a weekly or monthly basis, when the server would choose a new base key pair (PK, MK). The encryption of database records would be updated accordingly.

The corresponding modification of the definition of protocol π is straightforward.

7.2 The complexity of CRS setups

The use of some kind of trusted setup is unavoidable. In our protocol, the base type of setup is the Common Reference String (or Common Random String) setup. In single-session UC-security one-time CRS is used. However, it is unacceptably expensive in a practical application. We want to use a single CRS for an instance of our protocol with id sid, i.e. the same CRS for different sessions with identifier (sid, ssid). A reusable CRS-setup may serve such goal and fortunately there exist such construction, e.g. for construction in paper [9] for OT protocols and in paper [14] for zk_SNARK. Accordingly, we can use a composite reusable CRS-setup with components CRS_{OT} and CRS_{NIZK}.

The obvious related question is whether and how we can avoid completely relying on a single trusted party. The issue is further amplified by the fact that in known zk-SNARKs, one has to generate a new CRS each time the relation changes (note, different provers can prove the knowledge of their witness, provided that it is about the same relation). Accordinly, we need different CRSs for proving the honest computation of the commitments, the encryptions and the secret function keys in our protocol. It follows that CRS_{NIZK} means a triplet of CRSs: CRS_{NIZK_COM}, CRS_{NIZK_ENC} and CRS_{NIZK_SFK}.

For relaxing the trust in the setup several techniques were proposed. In the model proposed in [12] a number of users can update a universal CRS. The updatable CRS model guarantees security if at least one of the users updating the CRS is honest. In paper [1] UC-secure secure multiparty computation protocol is proposed for the generation of Common Random String setup. This allows to safely compose the CRS-generation protocol with the zk-SNARK in a black-box manner with the insurance that the security of the zk-SNARK is not influenced. This CRS-generation protocol also avoids the random oracle model which is typically not required by zk-SNARKs themselves. They assume the Registered Public Key (RK) setup model (a kind of public key distribution model). Recall by this model, an honest party can trust only its own key to be safe (i.e. trusts it own certification authority), i.e. the corresponding secret key is not leaked and other, even maliciously generated (public) keys are assumed to be well-formed. This latter assumption is consider as the weakened trust asssumption that holds with respect to other certification authorities.

7.3 Complexity of random oracle setup

Random oracles cannot be implemented efficiently. An ad hoc implementation is secure hash function (believed as a practical implementation of pseudorandom function). Following this (ad-hoc) logic an assumption of independent random oracle instance per protocol instance (with identifier sid) is intuitively impractical as a large number of different secure hash functions do not exist. It seems that the only solution is to assume that different instances of protocol π' have shared access to a global functionality, i.e. we consider global RO models. In this case, only a single secure hash function is required for an ad-hoc implementation of the global random oracle. Considering the programmable random GUC-oracle model defined in [7], by our claim Th.3. there exists FE scheme that is adaptively secure in this model. Designing such schemes with practical efficiency would be a fundamental step in the practical applicability of the protocol for the task of controlled private function evaluation in general (adaptive) application scenarios.

8. References

[1] Abdolmaleki, B. et. al. (2019),'UC-Secure CRS Generation for SNARKs', Progress in Cryptology, AFRICACRYPT 2019, Lecture Notes in Computer Science, Publisher Springer, Cham, Volume 11627, pp. 99-117

[2] Badertscher, C. et.al. (2021), 'Consistency for Functional Encryption', In 2021 IEEE 34th Computer Security Foundations Symposium (CSF). IEEE, 34th IEEE Computer Security Foundations Symposium, pp. 1-16

[3] Boneh, D., Sahai, A. and Waters, B. (2011),'Functional Encryption: Definitions and Challenges', Proceedings of Theory of Cryptography Conference (TCC) 2011, pp 253-273

[4] Canetti, R. (2002), 'Universally Composable Security: A New Paradigm for Cryptographic Protocols', In 34th STOC, pages 494_503, 2002.

[5] Canetti, R. et. al. (2002),'Universally composable two-party and multi-party secure computation', In 34th ACM STOC, Montreal, Quebec, Canada, ACM Press, pp. 494–503

[6] Camenisch, J., Dubovitskaya,M. and Neven, G., (2009), ‘Oblivious Transfer with Access Control’, 16th ACM Conference on Computer and Communications Security (ACM CCS 2009), pp. 131-140

[7] Camenisch, J. et al. (2018), ’The Wonderful World of Global Random Oracles’, in Advances in Cryptology – EUROCRYPT 2018, pp 280-312

[8] Cattaneo, G. et. al. (2020). ‘The European data market monitoring tool, Key facts & figures, first policy conclusions, data landscape and quantified stories: d2.9 final study report’. Publications Office of the EU, July 2020.

[9] Choi, S.G. et. al. (2014), ‘Efficient, Adaptively Secure, and Composable Oblivious Transfer with a Single, Global CRS’, https://eprint.iacr.org/2012/700.pdf

[10] Gorbunov, S., Vaikuntanathan, V. and Wee, H. (2012). ‘Functional Encryption with Bounded Collusions via Multi-Party Computation’, Advances in Cryptology – CRYPTO 2012, pp 162-179

[11] Groth, J. et.al. (2018), ’Updatable and Universal Common Reference Strings with Applications to zk-SNARKs’, CRYPTO 2018, LNCS 10993, pp. 698–728, 2018.

[12] Horvath, M. et. al (2019). ’There Is Always an Exception: Controlling Partial Information Leakage in Secure Computation’, Cryptology ePrint Archive: Report 2019/1302

[13] Kim, J., Lee, J. and Oh, H. (2020), ‘Simulation-Extractable zk-SNARK with a Single Verification’, IEEE Xplore, Volume 8, pp. 156569 – 156581

[14] Kosba A. et.al. (215), ‘How to Use SNARKs in Universally Composable Protocols’, Cryptology ePrint Archive, Report 2015/1093

[15] Lindell, Y. (2013) ‘Highly-Efficient Universally-Composable Commitments based on the DDH Assumption’, In EUROCRYPT 2011, pp. 446-466

[16] Matt, C. and Maurer, U. (2015), ‘A definitional framework for functional encryption’. In IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015, pp. 217–231

[17] Nielsen, J. B. (2002), ‘“Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case’, in Advances in Cryptology — CRYPTO 2002, ser. Lecture Notes in Computer Science, M. Yung, Ed. Springer Berlin Heidelberg, 2002, vol. 2442, pp. 111–126

[18] Otto, B. et. al. (2019), ‘Data ecosystems. Conceptual foundations, constituents and recommendations for action’, Technical report, Fraunhofer Institute for Software and Systems Engineering ISST, 10 2019.