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 [11]. 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 non-adaptive FE-setting secure realization of the ideal functionality is achievable in the standard model, otherwise, accessibility of random oracle is required.

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

1. Introduction

The growing importance of data is beyond question today [8], [17]. 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 $f \in F_B \backslash F_A$, else abort, [11]. 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 ($f, f \in F_B \backslash F_A$). 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],[15]. 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 [16] 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 const-ruction 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],[15]) 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-SNARKs (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 [11] is (naturally) the closest to our work. The MPC tasks considered here were introduced in [11]. Their definition of the tasks (their ideal functionality) was as given in the introduction above. Work [11] 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.

Adversarial model: We consider static malicious corruption adversary detailed as follows. The assumed goals of a malicious server include cheating with function key (when the function key provided by the server does not match the request (function f of the client) as well as attempts to learn the private input (f) of the client (i.e. breaking its privacy).

However, we assume that the integrity of the database (i.e. input $(x_1,\ldots,x_n)$) is guaranteed. Note, allowing a real malicious adversary that can modify 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. Henceforth we assume that the task of database integrity is separated from the task (PFE) we aim for. We can think equivalently that we assume a weaker real adversary that has no incentive to attack the integrity of the records or the integrity is guaranteed by assumption. The first case might be an acceptable assumption if the adversary is the service provider itself. An example of the second guarantee can be a regular offline comparison of the records to a master copy.

In contrast, a corrupt client only performs a passive attack, i.e. attacks against privacy (note, such assumption corresponds to the standard assumption for the FE task [3],[10],[15]). 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 (e.g. oracles, CRS setups). These issues will be detailed later (see in Section 5).

4. Non-adaptive case

First, we consider a base (one-time) 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. 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 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 unified into a single request message.

4.1 Base (one-time) ideal functionality

The functionality is shown in Figure 1.

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 (***InicClient***, 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 (***InicClient***, 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, then it ignores the input, else it 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 party S and aborts the session.

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

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

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

Upon receiving *(**Corrupt,*** sid, S*)* from the adversary and an input (sid, $x_1,\ldots,x_n$, F_A) has already been received from S, disclose this input to the adversary.

Figure 1: Base F_{rCPFE} ideal functionality

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.

An adversary may modify the input of a corrupted client. However, the integrity of the database (i.e. of input ($x_1,\ldots,x_n$)) is guaranteed. Recall, in our adversarial model we assumed a semi-honest behavior by a corrupted party C. This means that such a party will output ($f(x_1),\ldots,f(x_n)$) for an input f.

Input ($x_1,\ldots,x_n$) means the entire database content in the above definition. 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 base functionality

Protocol π builds on a q-NA-SIM-secure FE primitive, furthermore on a perfect hiding commitment, an oblivious transfer (OT), and a Non-Interactive Zero-Knowledge (NIZK) sub-protocol. Because the security analysis of the protocol will be modular (we use the technique of hybrid protocols), these sub-protocols must have appropriate simulation-based security (detailed subsequently).

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, the 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 commit to their inputs. We assume that the commitment mapping Comm is computationally binding and perfectly hiding. The steps follow:

Random element generation and commitment: Parties (S, C) (and repeated for parties (C, S)) realize the following task:

$(1^n, 1^n) \rightarrow ((r, r''), Comm(r, r''))$,

where party S commits to a uniform string r and commitment mapping Comm uses random value r" for randomization of the mapping. Furthermore (r, r") is guaranteed to be indistinguishable from a coin-tossing sequence. Party C receives commitment Comm(r, r") and party S receives random value r (to be used as its local random tape) as well as decommitment value r" (to be used later for proving its "proper behavior").

Input commitment: Party S commits to its input $x=(x_1,\ldots,x_n)$:

$((x, r'), 1^n) \rightarrow (-, Comm(x; r'))$,

where r' is randomness used by S in the commitment.

Let the ideal functionality corresponding to this phase of commitments be denoted by F_{COMMS}. The (F_{COMMS}, F_{OT}, F_{NIZK})-hybrid protocol π is defined in Figure 2. 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 ideal functionality F_{COMMS}

The main part of the protocol:

1. C → S: F_B ; 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_{OT}: f, f∈F

4. S → F_{NIZK}: sk_{fi} , i=1,…,m, $\{f_1,\ldots,f_m\} = F_B \setminus F_A$

5. F_{NIZK} → S: $Proof1_{fi}$, i=1,…,m

6. S → F_{OT}: SK=[(sk_{f1}, $Proof1_{f1}$), … , (sk_{fm}, $Proof1_{fm}$)], $\{f_1,\ldots,f_m\} = F_B \setminus F_A$

7. F_{OT} → C: sk_f, $Proof1_f$, if f∈ $F_B \setminus F_A$

8. F_{OT} → S: Abort3, if f∉ $F_B \setminus F_A$

9. C: verifies Proof1; if it does not verify, then C outputs message Abort4 and halts

10. S → F_{NIZK}: $E(x_1),\ldots,E(x_n)$

11. F_{NIZK} → S: Proof2

12. S → C: ENC=[$E(x_1),\ldots,E(x_n)$, Proof2]

13. C: verifies Proof2; if it does not verify, then C outputs message Abort5 and halts

14. C: outputs $D_{skf}(E(x_1)),\ldots, D_{skf}(E(x_n))$

Figure 2: The (F_{COMMS}, F_{OT}, F_{NIZK})-hybrid protocol π

Here, Proof1 and Proof2 are NIZK proofs. Their role is to enforce the corrupt server to generate function keys and ciphertexts, respectively, in an honest way. We notice that although party C only checks the honest generation of function key (sk_f) he is currently requesting (see Step 9), in the view of a corrupt server each secret key (sk_{fi} , i=1,…,m) is expected to be checked with a non-negligible probability. Because of this, a rational corrupt server will behave in a semi-honest manner. (See also the analysis of the simulation.)

By Proof2 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. Similarly, by Proof1, party S proves that function keys are

computed honestly, and accordingly when party C uses function key sk_f an encrypted record E(z), ∀z∈X (input space X) decodes into f(z).
Our claim about the security of hybrid protocol π is as follows:

Theorem 1: The (F_{COMMS}, F_{OT}, F_{NIZK})-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 COMMS, OT, and NIZK ideal subroutines (against a static, malicious adversary). The claim remains true even for a realization of these subroutines with a weaker security guarantee, more precisely when the realizations are simulation-based secure in a non-concurrent setting.

Proof: On the one hand we refer to the universal composition theorem [4], [5]. On the other, we notice that in the base scenario instances of the sub-protocols are called sequentially and each only with a single input (four instances are called from Comm, two from NIZK sub-protocols, and a single instance from the OT sub-protocol) ■

4.3 Analysis of the (F_{COMMS}, F_{OT}, F_{NIZK}) -hybrid protocol

We show that for arbitrary efficient adversary and arbitrary probability distribution p(.) over the input variable f, f∈ $F_B \backslash F_A$ of the client, there exists an efficient straight-line simulator Sim.

4.3.1 Definition of the simulator

Simulator Sim has access to the FE-simulator. In particular, it can run the key generation (*KeyGen*: $1^n \rightarrow$(PK, SK)), the function key simulation (*func_KeyGen*: f $\rightarrow f_{sk^*}$, f∈F) and the ciphertext simulation (*Encrypt*: x → c*, x∈X) algorithms of the FE-simulator. Simulator Sim simulates ideal functionalities F_{COMMS}, F_{OT}, and F_{NIZK} 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

As a preliminary step 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.

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.

Step 6. S→Sim(F_{OT}):

Party S sends message SK to F_{OT} via internal simulated interaction. Sim captures the message. Sim takes a sample f* from input distribution p(.). Sim verifies proof $Proof1_{f^*}$. If it does not verify, Sim sends message Abort4 to F_{rCPFE}. In return, functionality F_{rCPFE} sends message Abort4 as output to C.

Step 12. S→Sim(C):

Party S sends message ENC to C via internal simulated interaction. Sim captures the message and verifies the proof. If it verifies, then Sim extracts $x=(x_1,\ldots x_q)$ from Proof_1 and sends x as input to ideal functionality F_{rCPFE} on behalf of S via external interaction. Upon the arrival of input x, F_{rCPFE} sends output f(x) to party C. Otherwise, Sim sends message Abort5 to F_{PFE} and halts. In return, functionality F_{PFE} sends message Abort5 as output to C. ■

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 by the functionality outputs to party C. Sim learns this output and forwards it to party C.

Step 3. C→Sim(OT):

Party C sends input f to OT via internal simulated interaction.

Step 7. Sim(OT)→C:

Sim simulates message [sk_f, (Proof1)], i.e. the output of functionality OT is simulated to party C, where

- function key is simulated by calling the *func_KeyGen* algorithm of the FE-simulator,
- NIZK proofs are simulated by using the NIZK-simulator.

If party C sends output message Abort4, Sim forwards this message to the output of C, and the simulation halts.

Step 12. Sim(S)→C:

Sim simulates message ENC for C on behalf of S, where

- ciphertexts are simulated by calling the *Encrypt* algorithm of the FE-simulator,

- NIZK proofs are simulated by using the NIZK-simulator.

If party C sends output message Abort5, Sim forwards this message to the output of C, and the simulation halts. ■

4.3.2 Proof of indistinguishability

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 at every step of the ideal and real system run. Therefore, it is satisfactory to examine the indistinguishability of the view 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:
Party S is corrupt: F_B
Party C is corrupt: F_A, simulated [sk_f, $Proof1_f$], simulated ENC=[$E(x_1)$,…,$E(x_n)$, Proof2]

Real system:
Party S is corrupt: F_B
Party C is corrupt: F_A, [sk_f, $Proof1_f$], ENC=[$E(x_1)$,…,$E(x_n)$, Proof2]

It follows that the indistinguishability of the views can be reduced to the corresponding property of the simulators, i.e. the FE-simulator and the NIZK-simulator. (Note, all these proofs verify as correct.) Formally, let random variables V=(X,Y) and V'=(X',Y') correspond to the message views in the real and the ideal systems, respectively, where random variables X and Y correspond to the FE and the NIZK components, respectively. We can assume that random variables X and Y, similarly X' and Y' are statistically independent (based on independent random elements). Under this assumption we get

$d(V,V') \leq \min\{d(X,X'), d(Y,Y')\}$,

from which it follows that (computational) distance d(V,V') is also negligible.

Case of abort: We have to show the indistinguishability of abort probabilities in the two systems. Concretely, we show that corresponding probabilities of aborts are equal in the two systems. This is obvious for abort events due to erroneous inputs (Abort1-3). In the real system, abort events Abort4 and Abort5 happen if Proof1 and Proof2 do not verify as correct, respectively. Here follows that as long as a corrupt server behaves rationally and generates cheating proofs with only a negligible probability the ideal and real system become indistinguishable by the probabilities of abort events. ■

4.4 Parallelized version

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

The formal definition of the corresponding version of ideal functionality F_{rCPFE} is the same as the base functionality with a few (natural) differences. Function input f of the client is replaced by a sequence $f^*=(f_1,\ldots,f_q)$ of functions, where $f_i \in F$, $i=1,\ldots,q$ and q is fixed a priori. The OT sub-protocol processes the elements of the input sequence f* serially, i.e. one by one. Similarly, the necessary changes to the hybrid protocol (Figure 2.) are straightforward, so from space-saving reasons, those are not detailed here.

The technical details of the analysis are essentially the same as for the base case. Theorem 1 extends to this parallelized version.

5. Adaptive case

In a general application scenario, 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 as well as 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. An assumption on the availability of a public random oracle is necessary.

5.1 Ideal functionality F_{rCPFE}

Definition of (general) ideal functionality F'_{rCPFE} is shown in Figure 3. At first glance, this looks the same as the definition of the one-time functionality. The difference is that by the (general) 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 it to a fresh instance of communication keys (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, disclose this input to the adversary.

Figure 3: Ideal functionality F'_{rCPFE}

5.2 Realization of ideal functionality F'_{rCPFE}

Because we want UC-security, a realization of each subtask (FE, OT, COMMS, NIZK) requires an appropriate trust-based setup. Sub-protocols commitment, NIZK, and OT can UC-securely be implemented in their respective CRS model. Additionally, each protocol instance uses its own independent random oracle instance, called the local ROM model. 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_{COMMS}, F_{OT}, F_{NIZK}, F_{RO} –hybrid protocol be denoted π'.

Note, that a CRS setup can be implemented via a trusted third party (TTP) or via a multi-party protocol. The latter solution either cannot completely eliminate some trust assumptions however it can substantially alleviate them. Subsequently, we briefly review also the trust assumptions for each subtask.

The analysis goes in F_{COMMS}, F_{OT}, F_{NIZK}, F_{RO} –hybrid. The key technical points of the simulation and its analysis are essentially the same as in the base case, therefore we do not detail it here due to space-saving. Our claim is as follows.

Theorem 2: (F_{COMMS}, F_{OT}, F_{NIZK}, 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: It is essentially similar to the proof of Theorem 1 with the following difference. The FE simulator needs to access a 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, assuming UC-secure realization of COMMS, OT, and NIZK ideal subroutines (against a static, malicious adversary).

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

In the remainder of this article, we provide an overview of the feasibility of the components of protocol π'.

5.2.2 FE primitive

The main difficulty with the realization of 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}.

Assuming independent random oracle instance per protocol instance is even intuitively highly impractical as a large number of different secure hash functions (believed as a practical implementation of pseudorandom function) do not exist. Therefore, 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. Several global RO models were proposed in [7].

However, there is a big technical obstacle here. 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 proof-technical problem is the following. 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 that the (classic) 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). (Recall that we need a straight-line simulator when we prove UC-security.) Our point here is that it turns out 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 in programmable random GUC-oracle setting.*

Proof: 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) can be adapted. ■

5.2.3 NIZK subprotocol

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 both kinds of NIZK-proofs (Proof1 and Proof2) used in our protocol. We need UC-secure Zk-SNARK [12], [13].

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. Subverting the setup phase can make it easy to break the security, e.g., leaking a CRS trapdoor makes it trivial to prove false statements.

A solution to avoid such a third party is the generation of CRS via a multi-party protocol. This way instead of assuming a TTP, parties can run a sub-protocol under minimized trust assumptions. For our purposes, such a sub-protocol has to achieve UC-security in order to safely compose the CRS-generation protocol with the Zk-SNARK in a black-box manner (i.e. with the insurance that the security of the Zk-SNARK is not influenced). Fortunately, this problem has also been successfully solved [1].

5.2.4 COMMS and OT subprotocols

COMMS subtask contains four commitment steps (one for inputs and another for random elements by both parties). Several UC-secure commitment constructions are known. There are works, where the CRS is one-time, i.e., one needs a new common-reference string for each execution of the commitment protocol. In other works, the CRS is independent of the number of parties and re-usable. E.g. the UC-secure commitment in [14] against a static corruption adversary requires a CRS that can be used by any number of parties, i.e. a single CRS can be published for all to use.

In [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.

Finally, we make a practical prediction on setup implementations. Predictably, tamperproof hardware token-based distributed setups could bring significant practical breakthroughs for efficient provably UC-secure implementations in the coming years. Mass application of cryptographic protocols could significantly accelerate this process.

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

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

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

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

[15] 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

[16] 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

[17] 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.