An extended abstract of this paper is published in the proceedings of the 16th International Conference on Practice and Theory of Public-Key Cryptography—PKC 2013. This is the full version.

Rate-Limited Secure Function Evaluation

Özgür Dagdelen[1], Payman Mohassel[2], and Daniele Venturi[3]

[1] *bridgingIT, N7 5, 68161 Mannheim, Germany*
[2] *Department of Computer Science, University of Calgary, Canada*
[3] *Department of Computer Science, Sapienza University of Rome, Italy*

March 10, 2016

Abstract

We introduce the notion of rate-limited secure function evaluation (RL-SFE). Loosely speaking, in an RL-SFE protocol participants can monitor and limit the number of distinct inputs (i.e., *rate*) used by their counterparts in multiple executions of an SFE, in a private and verifiable manner. The need for RL-SFE naturally arises in a variety of scenarios: e.g., it enables service providers to "meter" their customers' usage without compromising their privacy, or can be used to prevent oracle attacks against SFE constructions.

We consider three variants of RL-SFE providing different levels of security. As a stepping stone, we also formalize the notion of commit-first SFE (CF-SFE) wherein parties are committed to their inputs before each SFE execution. We provide compilers for transforming any CF-SFE protocol into each of the three RL-SFE variants. Our compilers are accompanied with simulation-based proofs of security in the standard model and show a clear tradeoff between the level of security offered and the overhead required. Moreover, motivated by the fact that in many client-server applications clients do not keep state, we also describe a general approach for transforming the resulting RL-SFE protocols into *stateless* ones.

As a case study, we take a closer look at the oblivious polynomial evaluation (OPE) protocol of Hazay and Lindell, show that it is commit-first, and instantiate efficient rate-limited variants of it.

Keywords: secure function evaluation; secure metering; oracle attacks.

Contents

1 Introduction

Secure function evaluation (SFE) allows a set of mutually distrustful parties to securely compute a function f of their private inputs. Roughly speaking, SFE protocols guarantee that the function is computed correctly and that the parties will not learn any information from the interaction, other than their output and what is inherently leaked from it. Seminal results in SFE show that one can securely compute any functionality [Yao82, Yao86, GMW87, BG89, CCD88]. There has been a large number of follow-up work improving the security, strengthening adversarial models, and studying efficiency. Recent work on practical SFE has also led to real-world deployments [BCD$^+$09, BTW12], and the design and implementation of several SFE frameworks [MNPS04, BLW08, DGKN09, HKS$^+$10, HEKM11].

In practice, most applications of SFE considered in the literature need to accommodate *multiple executions* of a protocol. Depending on the application, a subset of the participants may use the same input in different executions. Consider a client that searches for multiple patterns in a large text via a secure pattern matching protocol [HL08, HT10], searches several keywords in a private database via an oblivious keyword search [OK04, FIPR05], or an individual who needs to run a software diagnostic program, or an intrusion detection system (IDS) to analyze data via an oblivious branching program (OBP) or an oblivious automaton evaluation (OAE) protocol [IP07, TPKC07].

Invoking an SFE protocol multiple times raises important practical issues that are outside the scope of standard SFE, and hence are not addressed by the existing solutions. We point out two such issues below, and introduce *rate-limited* SFE as a means to address them. The reason for the choice of name is that rate-limiting is commonly used in network and web applications to refer to restrictions put on clients' usage (on a per user, or a per IP address basis). In this work we consider similar restrictions on a user's inputs to services that may be implemented using SFE.

Secure metering of SFE. Service providers tend to charge their clients according to their level of usage. Consider, for instance, a location-based service, where users are charged based on the number of locations they use the service from. On the one hand, there must be a limit on the number of locations queried by each user, in order to prevent users from learning too much information and replicating the service; on the other hand, users might care about hiding how many different locations they have accessed (in order to hide their speed).

Similar rate-limited settings include: A database owner charging clients based on the number of distinct search queries, and an IDS provider charging customers based on the number of suspicious files sent for vulnerability analysis. Service providers would be more willing to adopt SFE protocols if it is possible to efficiently enforce such a metering mechanism. The challenge is to do so without compromising the client's privacy, or allowing the server or the client to cheat the metering system.

Oracle attacks. Consider multiple executions of a two-party SFE protocol (such as those mentioned above), where the first party's input stays the same in different executions but the second party's input varies. A malicious second party who "adaptively" uses different inputs in each execution, can gradually learn significant information about the first party's input, and, in the worst case, fully recover it. For instance, consider an oblivious polynomial evaluation (OPE) protocol (e.g., used in oblivious keyword search) wherein the server holds a polynomial p while the client holds a private point x and wants to learn $p(x)$, but cannot learn more than that. Evaluating the polynomial p on sufficiently many points allows a malicious client to interpolate and recover p. A similar attack can be applied to OBP and OAE protocols to learn the private branching program or automaton which may embed propriety information. Learning attacks of this sort are well-understood and have been previously identified as important threats in the context of SFE; they are sometimes referred to as *oracle attacks* since the attacker has black-box access to input/output values from multiple executions (e.g., see the discussion in [BFK$^+$09]).

A naïve solution to the problems discussed above is to limit the total number of executions of an SFE protocol, ignoring the actual input values. However, this approach does not provide a satisfactory solution in most scenarios. For example, in case of secure metering, fixing an a priori upper bound on the total number of executions would mean charging legitimate clients multiple times for using the service with the same input; a disadvantage for clients who may need to use the same input from multiple devices, or reproduce a result due to communication errors, device upgrades, or perhaps to prove the validity of the outcome to a third-party by re-running the protocol. Similarly, in case of oracle attacks, clients need not be disallowed to use the same input multiple times since querying the same input many times does not yield new information to an attacker.

Rate-limited SFE. A more accurate (and challenging) solution is to *limit* and/or *monitor* the number of distinct inputs used by an SFE participant in multiple executions. Obviously, this should be done in a secure and efficient manner, i.e., a party should not be able to exceed an agreed-upon limit, and its counterpart should not learn any additional information about his private inputs, or impose a lower limit than the one they agreed on.[1] We refer to the number of distinct inputs used by a participant as his *rate*, and call an SFE protocol that monitors/limits this number, a rate-limited SFE.

Of course, achieving RL-SFE is more costly than the naïve solution discussed above. However, at a minimum we require the proposed solutions to avoid storing and/or processing the complete transcripts of all previous executions. (We discuss the exact overhead of our solutions in detail below.)

We note that the complementary question of what functions are *unsafe* for use in SFE (leak too much information) has also been studied, e.g., by combining SFE and differential privacy [BNO08, MMP$^+$10], or belief tracking techniques [MHKS12]. These works are orthogonal to ours, and can potentially be used in conjunction with rate-limited SFE as an *enforcement mechanism*. For instance, the former works can be invoked to negotiate on a function f with a measurable "safeness" from which the rate for each user can be derived. Subsequently, the abidance of this rate can be enforced through our rate-limited SFE.

1.1 Our Contribution

Motivated by the discussion above, we initiate the study of rate-limited SFE in the two-party setting.

Definitions. We introduce three definitions for rate-limited secure function evaluation: (i) rate-hiding, (ii) rate-revealing and (iii) pattern-revealing. All our definitions are in the real-world/ideal-world simulation paradigm and are concerned with *multiple* sequential executions of an SFE protocol. They reduce to the standard (stand-alone) simulation-based definition for SFE, under static corruptions, when applied to a single execution.

In a *rate-hiding* RL-SFE, in each execution, the only information revealed to the parties is whether the agreed-upon rate limit has been exceeded or not. In a *rate-revealing* RL-SFE,

[1] Note that although the problem becomes trivial when the parties are assumed to be semi-honest, it is already interesting in the augmented semi-honest setting [Gol09, Chapter 7], where the parties are allowed to modify their inputs in each protocol execution.

the parties additionally learn the current rate (i.e., the number of distinct inputs used by their counterpart so far). In a *pattern-revealing* RL-SFE, parties also learn the pattern of occurrences of each other's inputs during the previous executions. These notions provide a useful spectrum of tradeoffs between security and efficiency: our constructions become more efficient as we move to the more relaxed notions, to the extent that *our pattern-revealing transformation roughly adds no overhead* to the underlying SFE protocol.

Commit-first SFE. In order to design rate-limited SFE protocols, we formalize the auxiliary notion of commit-first SFE (CF-SFE). Roughly speaking, a protocol is commit-first if it can be naturally divided into a (i) *committing phase*, where each party becomes committed to its input for the second phase, and (ii) a *function evaluation phase*, where the function f is computed on the inputs committed to in the first phase.[2] The notion of CF-SFE is related to the well understood principle of "evaluating on committed inputs" (cf. Section 1.2).

We utilize CF-SFE as a stepping stone to design rate-limited SFE. It turns out that the separation between the input commitment phase and the function evaluation phase facilitates the design of efficient rate-limited SFE. In particular, now a party only needs to provide some evidence of a particular relation between the committed inputs in the first phase. In contrast, if we had not started with a commit-first protocol, such an argument would have involved the complete history of the transcripts for all the previous executions, rendering such an approach impractical.

In order to prove our RL-SFE protocols secure, we put forth a formal and general definition for CF-SFE. We then show that several existing SFE constructions are either commit-first or can be efficiently transformed into one. Examples include variants of Yao's garbled circuit protocol, the oblivious polynomial evaluation of Hazay and Lindell [HL09], the private set intersection protocol of Hazay and Nissim [HN12], and the oblivious automaton evaluation of Gennaro *et al.* [GHS10]. We also show that the GMW compiler [GMW87] outputs a commit-first protocol. This is of theoretical interest as it provides a general compiler for transforming a semi-honest SFE protocol into a malicious CF-SFE (and eventually a rate-limited SFE using the compilers in this paper). We elaborate on these CF-SFE instantiations in Section 3.2.

Compilers & techniques. We design three compilers for transforming a CF-SFE into each of the three variants of RL-SFE discussed above, and provide simulation-based proofs of their security. All our compilers start from a CF-SFE protocol and add a "proof of repeated-input phase" between the committing phase and the function evaluation phase. An exception is our pattern-revealing compiler, where a proof of repeated-input is implicit given that we force the commitments to be deterministic. In our first compiler (rate-hiding), whenever the j-th execution begins, party P_1 first checks whether its input is "fresh" or has already been used in a previous run. In the former case, P_1 encrypts the value "1" and, otherwise, the value "0" using a semantically secure public-key encryption scheme (E, D) for which it holds the secret key sk. Denote the resulting ciphertext with c^j. Party P_1 forwards to P_2 a zero knowledge (ZK) proof of the following statement:

$$\begin{aligned}&(\text{“committed to old input”} \wedge \text{“encryption of 0”})\\ &\qquad \vee (\text{“committed to new input”} \wedge \text{“encryption of 1”} \wedge \text{“}\textstyle\sum_{i\leq j} \mathsf{D}(sk, c^i) \leq \text{rate”}).\end{aligned}$$

Intuitively, the proof above only leaks the fact that the rate is not exceeded in the current execution, but nothing else. In order to generate this proof (resp. verify the proof generated by

[2] Note that adding input commitments to the beginning of a protocol does not automatically yield a CF-SFE, since parties are not necessarily bound to using the committed inputs in their evaluation.

the counterpart), P_1 needs to store all the commitments and ciphertexts sent to (resp. received from) P_2 in previous executions.

For our second compiler (rate-revealing), we can do without the encryptions. The parties can instead prove a simpler statement giving evidence that the current (committed) input corresponds to one of the commitments the other party received earlier. Clearly, this approach reveals the current rate, but as we prove nothing more.

Finally, our third compiler (pattern-revealing) exploits a PRF to generate the randomness used in the committing phase of the underlying CF-SFE protocol. In this way, the commitment becomes deterministic (given the input), allowing the other party to check whether the current input has already been used and *in which runs*. This approach discloses the pattern of inputs used by the parties; on the other hand, it is extremely efficient adding little computational overhead (merely one invocation of a PRF) to the original CF-SFE protocol.

Making RL-SFE stateless. The above compilers suffer from the limitation that the parties need to keep a state which grows linearly in the total number of executions of the underlying SFE protocol. In many applications, clients do not keep state (and outsource this task to the servers), either due to lack of resources or because they need to use the service from multiple locations/devices. We show a general approach for transforming the stateful RL-SFE protocols generated above into *stateless* ones. Here, the client keeps only a small secret (whose size is independent of the total number of executions), but is still able to prevent cheating by a malicious server, and preserve privacy of his inputs. At a high level, the transformation requires the client to store its *authenticated* (MACed) state information on the server side and retrieve/verify/update it on-the-fly as needed. We show how to apply this transformation to our rate-revealing compiler to obtain a stateless variant and prove its security. A similar technique can be applied to our rate-hiding compiler. Our pattern-revealing compiler is already stateless for the party who plays the role of the client (as the client only needs to store a PRF key).

Case study. We take a closer look at the oblivious polynomial evaluation (OPE) protocol of Hazay and Lindell [HL09]. Their protocol is secure against malicious adversaries. We show that it is also a commit-first OPE, by observing that a homomorphic encryption of the parties' inputs can be interpreted as a commitment to their inputs. This immediately yields an efficient pattern-revealing RL-SFE for the OPE problem, based on the compiler we design. We also provide an efficient rate-hiding and rate-revealing RL-OPE by instantiating the ZK proofs for membership in the necessary languages, efficiently.

1.2 Additional Related Work

Our notion of CF-SFE is related to the notion of "committed oblivious transfer" [CvdGT95, Gar04], and more generally of "evaluating on committed inputs" (see, e.g., [GMW87, JS07]); however, to the best of our knowledge, ours is the first general definition of this feature.

Conference version. An abridged version of this paper appeared as [DMV13]. This is the full version containing new material and significantly revised proofs. In particular, our compilers for RL-SFE, the instantiations of CF-SFE, and the case study on OPE, were described and analyzed only in a very high-level manner in the proceeding version (while they are treated in full details here).

Subsequent to our work, RL-SFE has been suggested as a useful tool in different contexts, including secure cloud storage [ADDV15, ADDV16], pattern matching [FHV13], and cooperative linear algebra [DV14].

1.3 Roadmap

We discuss some preliminaries in Section 2 and give our model for commit-first SFE in Section 3. The definition of rate-limited SFE is introduced in Section 4. Our rate-hiding, rate-revealing and pattern-revealing compilers are described and analyzed in Section 5, whereas Section 6 describes the stateless version of the rate-revealing compiler. Finally, Section 7 deals with concrete instantiations for the case of OPE, and Section 8 explains the main problems left open by our work.

2 Preliminaries

After setting some basic notation in Section 2.1, we review the definitions of the main cryptographic primitives on which we build in Section 2.2.

2.1 Notation

Throughout the paper, we denote the security parameter by $\lambda \in \mathbb{N}$. A function $\nu : \mathbb{N} \to [0, 1]$ is negligible in λ (or just negligible) if it decreases faster than the inverse of every polynomial in λ. A machine is said to be probabilistic polynomial-time (PPT) if it uses randomness as parts of its logic, and its number of steps is polynomial in the input size.

Let $X = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ and $Y = \{Y_\lambda\}_{\lambda\in\mathbb{N}}$ be two distribution ensembles. We say X and Y are computationally indistinguishable (and we write $X \equiv_c Y$) if for every non-uniform polynomial-time adversary $\mathcal{A}$ there exists a negligible function $\nu : \mathbb{N} \to [0, 1]$ such that $|\Pr[\mathcal{A}(X) = 1] - \Pr[\mathcal{A}(Y) = 1]| \le \nu(\lambda)$. Note that all our security statements can be straightforwardly proven for uniform polynomial-time adversaries, as well.

If x is a string, $|x|$ denotes the length of x. Vectors are denoted boldface; given vector $\mathbf{x}$, we write $\mathbf{x}[j]$ for the j-th element of $\mathbf{x}$. If $\mathcal{X}$ is a set, $\#\mathcal{X}$ represents the number of elements in $\mathcal{X}$. When x is chosen randomly in $\mathcal{X}$, we write $x \leftarrow \mathcal{X}$. When $\mathcal{A}$ is an algorithm, $y \leftarrow \mathcal{A}(x)$ denotes a run of $\mathcal{A}$ on input x and output y; if $\mathcal{A}$ is randomized, then y is a random variable and $\mathcal{A}(x; r)$ denotes a run of $\mathcal{A}$ on input x and random coins r.

2.2 Basic Cryptographic Tools

2.2.1 Commitment Schemes

A non-interactive commitment scheme is a randomized efficient algorithm C taking as input a message $m \in \mathcal{M}$ and random coins $r \in \mathcal{R}$, and outputting a commitment $\gamma \in \mathcal{C}$. A decommitment of γ consists simply of revealing m and r. The sets $\mathcal{M}$, $\mathcal{R}$ and $\mathcal{C}$ are called (respectively) the message space, the randomness space, and the commitment space. A commitment scheme satisfies two properties called *hiding* and *binding*. We recall such properties below.

The binding property says that it is hard to open a given commitment $\gamma \in \mathcal{C}$ in two different ways.

Definition 2.1 (Binding)**.** *We say that a non-interactive commitment* C *is* perfectly binding *if there do not exist pairs* $(m_0, r_0), (m_1, r_1)$ *such that* $m_0 \neq m_1$ *and, at the same time,* $\mathsf{C}(m_0; r_0) = \mathsf{C}(m_1; r_1)$*.*

We recall that, in the plain model, the assumption of perfect binding (instead of computationally binding) is without loss of generality.

The hiding property says that for any pair of messages m_0, m_1 it is hard to tell whether a given commitment γ is for m_0 or for m_1.

Definition 2.2 (Hiding)**.** *We say that a non-interactive commitment* C *is* computationally hiding *if for all messages* $m_0, m_1 \in \mathcal{M}$ *the following holds:*

$$\{\gamma : \ \gamma \leftarrow \mathsf{C}(m_0)\}_{\lambda \in \mathbb{N}} \equiv_c \{\gamma : \ \gamma \leftarrow \mathsf{C}(m_1)\}_{\lambda \in \mathbb{N}} .$$

Non-interactive commitment schemes exists based on explicit hardness assumptions, e.g. Pedersen's commitment [Ped92], and more in general from any one-way permutation [GL89].

2.2.2 Zero-Knowledge Arguments

A decision problem related to a language $L \subseteq \{0,1\}^*$ requires to determine if a given string x is in L or not. We can associate to any NP-language L a polynomial-time computable relation R defining L itself, that is $L = \{x : \exists w \text{ s.t. } R(x,w) = 1\}$, where $|w|$ is at most polynomial in $|x|$. The string w is called a witness for membership of $x \in L$.

An argument of membership (or simply an argument) for a given language L, is a possibly interactive protocol $(\mathcal{P}, \mathcal{V})$ between two parties where the prover $\mathcal{P}$ convinces the verifier $\mathcal{V}$ that some string x belongs to the language L at hand; the prover additionally holds a witness w for x, i.e. $R(x,w) = 1$. At the end of the protocol execution, the verifier outputs a bit (representing his decision); we write $\langle \mathcal{P}(w), \mathcal{V} \rangle(x)$ for the random variable corresponding to the verifier's verdict. Similarly, we write $\mathcal{P}(x,w) \leftrightarrows \mathcal{V}(x)$ for the random variable corresponding to transcripts of honest protocol executions. The prover and the verifier itself, constitute what is called a (possibly interactive) *argument system.*

An argument system should satisfy at least two properties, completeness and soundness. Completeness says that an honest prover (holding a valid witness) is able to convince the verifier.

Definition 2.3 (Completeness)**.** *Let* $(\mathcal{P}, \mathcal{V})$ *be an argument system for an NP-language* L *(with corresponding relation* R*). We say that* $(\mathcal{P}, \mathcal{V})$ *satisfies completeness if for all* (x, w) *such that* $R(x,w) = 1$ *there exists a negligible function* $\nu : \mathbb{N} \to [0,1]$ *such that*

$$\Pr\left[\langle \mathcal{P}(w), \mathcal{V} \rangle(x) = 1\right] \geq 1 - \nu(\lambda),$$

where the probability is taken over the randomness of algorithms $\mathcal{P}$ *and* $\mathcal{V}$*.*

Soundness informally says that, whenever $x \notin L$, no computationally bounded prover can convince the verifier into accepting x.

Definition 2.4 (Soundness)**.** *Let* $(\mathcal{P}, \mathcal{V})$ *be an argument system for an NP-language* L*. We say that* $(\mathcal{P}, \mathcal{V})$ *satisfies soundness if for all PPT algorithms* $\mathcal{P}^*$*, and for any* $x \notin L$*, there exists a negligible function* $\nu : \mathbb{N} \to [0,1]$ *such that*

$$\Pr\left[\langle \mathcal{P}^*, \mathcal{V} \rangle(x) = 1\right] \leq \nu(\lambda),$$

where the probability is taken over the randomness of algorithms $\mathcal{P}^*$ *and* $\mathcal{V}$*.*

Completeness and soundness do not quantify how much information an interactive argument reveals about the witness, which in turn can be covered by the zero-knowledge property defined below.

Definition 2.5 (Zero-Knowledge)**.** *Let* $(\mathcal{P}, \mathcal{V})$ *be an argument system for an NP-language* L *(with corresponding relation* R*). We say that* $(\mathcal{P}, \mathcal{V})$ *satisfies zero-knowledge if there exists a PPT simulator* $\mathcal{S}$ *such that for all PPT algorithms* $\mathcal{V}^*$*, for all* (x, w) *such that* $R(x,w) = 1$*, and for all auxiliary inputs* $z \in \{0,1\}^*$*, the following holds:*

$$\{\mathcal{P}(x,w) \leftrightarrows \mathcal{V}^*(x,z)\} \equiv_c \{\mathcal{S}^{\mathcal{V}^*}(x,z)\}.$$

An important result in the theory of zero-knowledge is that every language in NP admits a ZK argument system [GMW91]. In particular, if there exist argument systems for language L_1, L_2 in NP, then it is possible to prove arbitrary combinations of statements from the two languages, e.g., it is possible to prove statements of the form $(x_1 \in L_1) \wedge (x_2 \in L_2)$ and $(x_1 \in L_1) \vee (x_2 \in L_2)$.

2.2.3 Pseudo-Random Functions

Let $\mathsf{PRF} : \mathcal{K} \times \mathcal{M} \to \mathcal{N}$ be a function, where $\mathcal{K}$ represents the key space, $\mathcal{M}$ the message space, and $\mathcal{N}$ the output space. Roughly, PRF is a secure pseudo-random function (PRF) if, for a random key, it is indistinguishable from a truly random function mapping $\mathcal{M}$ into $\mathcal{N}$.

Definition 2.6 (PRF). *We say that* $\mathsf{PRF} : \mathcal{K} \times \mathcal{M} \to \mathcal{N}$ *is a PRF if for all PPT distinguishers* $\mathcal{D}$ *there exists a negligible function* $\nu : \mathbb{N} \to [0,1]$ *such that*

$$\left| \Pr[\mathcal{D}^{\mathsf{PRF}(k,\cdot)}(1^\lambda) = 1 : \; k \leftarrow \mathcal{K}] - \Pr[\mathcal{D}^{\mathsf{F}(\cdot)}(1^\lambda) = 1] \right| \leq \nu(\lambda),$$

where F *is chosen at random from the set of all functions mapping* $\mathcal{M}$ *into* $\mathcal{N}$*.*

2.2.4 Public-Key Encryption

A public-key encryption (PKE) scheme is a triple of efficient algorithms $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ defined as follows. Upon input the security parameter λ, the probabilistic algorithm G outputs a pair of keys (pk, sk). Upon input the key pk and message $m \in \mathcal{M}$, the probabilistic algorithm E outputs $c \leftarrow \mathsf{E}(pk, m)$ where c belongs to ciphertext space $\mathcal{E}$. Upon input the key sk and a ciphertext c, the deterministic algorithm D outputs a message $m \in \mathcal{M}$.

A PKE scheme is correct if for all $(pk, sk) \leftarrow \mathsf{G}(1^\lambda)$, and for all messages m, we have that $\mathsf{D}(sk, \mathsf{E}(pk, m)) = m$. The standard security notion for PKE schemes is called CPA-security, and is defined below.

Definition 2.7 (CPA Security for PKE). *We say that a PKE scheme* $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ *is CPA secure if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* $\nu : \mathbb{N} \to [0,1]$ *such that*

$$\Pr\left[b' = b : \begin{array}{c} (pk, sk) \leftarrow \mathsf{G}(1^\lambda); b \leftarrow \{0,1\}; (m_0, m_1, \tau) \leftarrow \mathcal{A}(pk) \\ c \leftarrow \mathsf{E}(pk, m_b); b' \leftarrow \mathcal{A}(pk, c, \tau) \end{array} \right] \leq \nu(\lambda),$$

where the probability is taken over the randomness of algorithms G, E*, and* $\mathcal{A}$*.*

A PKE scheme is (additively) homomorphic, if there exist operations $+_h$ and $-_h$ on the ciphertext space such that, given $c_1 \leftarrow \mathsf{E}(pk, m_1)$ and $c_2 \leftarrow \mathsf{E}(pk, m_2)$, computing $c_1 +_h c_2$ (resp., $c_1 -_h c_2$) results in an encryption of $m_1 + m_2$ (resp., $m_1 - m_2$).

2.2.5 Secret-Key Encryption

A secret-key encryption (SKE) scheme is a triple of efficient algorithms $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ specified as follows. Upon input the security parameter λ, the probabilistic algorithm G outputs a key $k \in \mathcal{K}$. Upon input the key k and message $m \in \mathcal{M}$, the deterministic algorithm E outputs $c = \mathsf{E}(k, m)$ where c belongs to ciphertext space $\mathcal{E}$. Upon input the key k and a ciphertext c, the deterministic algorithm D outputs a message $m \in \mathcal{M}$.

An SKE scheme is correct if for all $k \leftarrow \mathsf{G}(1^\lambda)$, and for all messages m, we have that $\mathsf{D}(k, \mathsf{E}(k, m)) = m$. The standard security notion for SKE schemes is called CPA-security, and is defined below.

Definition 2.8 (CPA Security for SKE)**.** *We say that an SKE scheme* $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ *is CPA secure if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* $\nu : \mathbb{N} \to [0, 1]$ *such that*

$$\Pr\left[b' = b : \begin{array}{c} k \leftarrow \mathsf{G}(1^\lambda); b \leftarrow \{0,1\}; (m_0, m_1, \tau) \leftarrow \mathcal{A}^{\mathsf{E}(k,\cdot)}(1^\lambda) \\ c = \mathsf{E}(k, m_b); b' \leftarrow \mathcal{A}(c, \tau) \end{array}\right] \leq \nu(\lambda),$$

where the probability is taken over the randomness of algorithms G *and* $\mathcal{A}$*.*

2.2.6 Collision-resistant Hashing

We recall what it means for a family of hash functions to be collision resistant. Let $l, l' : \mathbb{N} \to \mathbb{N}$ be such that $l(\lambda) > l'(\lambda)$, and let $I \subseteq \{0,1\}^*$.

Definition 2.9 (Collision-Resistant Hashing)**.** *A function family* $\{H_\iota\}_{\iota \in I}$ *is called a collision-resistant hash family if the following holds.*

- *There exists a probabilistic polynomial-time algorithm* I *that on input* 1^λ *outputs* $\iota \in I$*, indexing a function* H_ι *mapping from* $l(\lambda)$ *bits to* $l'(\lambda)$ *bits.*
- *There exists a deterministic polynomial-time algorithm that on input* $x \in \{0,1\}^l$ *and* $\iota \in I$*, outputs* $H_\iota(x)$*.*
- *For all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* $\nu : \mathbb{N} \to [0, 1]$ *such that*

  $$\Pr\left[H_\iota(x) = H_\iota(x') : (x, x') \leftarrow \mathcal{A}(1^\lambda, \iota); \iota \leftarrow \mathsf{I}(1^\lambda)\right] \leq \nu(\lambda),$$

  where the probability is taken over the coin tosses of algorithms I *and* $\mathcal{A}$*.*

2.2.7 Message Authentication Codes

A message authentication code (MAC) $(\mathsf{S}, \mathsf{T}, \mathsf{V})$ is a triple of efficient algorithms specified as follows. Upon input security parameter λ, the probabilistic setup algorithm S outputs a key $s \in \{0,1\}^\kappa$. Upon input the key s and a message $m \in \mathcal{M}$, the deterministic algorithm T outputs a tag $\phi = \mathsf{T}(s, m)$. Upon input the key s and a pair (m, ϕ), the deterministic algorithm V returns 1 if and only if $\phi = \mathsf{T}(s, m)$. Correctness requires that for any $s \leftarrow \mathsf{S}(1^\lambda)$, and any message $m \in \mathcal{M}$, we have that $\mathsf{V}(s, m, \mathsf{T}(s, m))$ outputs 1.

The standard security notion for MACs is called existential unforgeability under chosen-message attacks (EUF-CMA), and is given below.

Definition 2.10. *We say that a MAC* $(\mathsf{S}, \mathsf{T}, \mathsf{V})$ *is EUF-CMA if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* $\nu : \mathbb{N} \to [0, 1]$ *such that*

$$\Pr\left[\mathsf{V}(s, m^*, \phi^*) = 1 \wedge m^* \notin \mathcal{Q} : \quad s \leftarrow \mathsf{S}(1^\lambda); (m^*, \phi^*) \leftarrow \mathcal{A}^{\mathsf{T}(s,\cdot)}(1^\lambda) \ \right] \leq \nu(\lambda),$$

where $\mathcal{Q}$ *contains the list of all messages asked to the* T *oracle, and where the probability is taken over the randomness of algorithms* S *and* $\mathcal{A}$*.*

3 Commit-First SFE

3.1 The Definition

In this section, we formally define the notion of *commit-first secure function evaluation* (CF-SFE). Our compilers for designing rate-limited SFE, leverage commit-first protocols as a building block. We call a protocol π commit-first (w.r.t. a pair of commitment schemes $\mathsf{C}_1, \mathsf{C}_2$) if it can be naturally divided into two phases. In the first phase (committing phase), both parties P_1 and P_2 become committed to their inputs (using C_1 and C_2, respectively). At the end of this phase, no information about the parties' inputs is revealed (the hiding property), and neither party can use a different input than what it is committed to in the remainder of the protocol (the binding property). In the second phase (function evaluation phase), the function f will be computed on the inputs committed to in the first phase.

We now describe the two separate phases more precisely. Consider a polynomial-time functionality $f = (f_1, f_2)$ with $f_i : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$. Then, a CF-SFE protocol $\pi = (\pi_1, \pi_2)$ for evaluating f on parties' inputs x_1 and x_2 proceeds as follows.

Committing Phase: Parties P_1 and P_2 execute π_1 which is defined by the functionality $((x_1, r_1), (x_2, r_2)) \mapsto ((x_1, r_1, \mathsf{C}_2(x_2; r_2)), (x_2, r_2, \mathsf{C}_1(x_1; r_1)))$. Note that the commitments $\mathsf{C}_1, \mathsf{C}_2$ (cf. Section 2.2) can be arbitrary schemes (often different for each CF-SFE protocol), as long as they satisfy the required hiding and binding properties.

Function Evaluation Phase: Afterwards, P_1 and P_2 execute π_2 on the same inputs as in the committing phase; π_2 is defined by the functionality $((x_1, r_1, \mathsf{C}_2(x_2; r_2)), (x_2, r_2, \mathsf{C}_1(x_1; r_1))) \mapsto (f_1(x_1, x_2), f_2(x_1, x_2))$. Note that P_1 and P_2, can use their state information from the previous phase in the function evaluation phase, too.

Next, we formalize the security definition for a CF-SFE using the real/ideal world simulation paradigm.

The real world. In each execution, a non-uniform adversary $\mathcal{A}$ following an arbitrary efficient strategy can send messages in place of the corrupted party (whereas the honest party continues to follow π). Let $i \in \{1, 2\}$ be the index of the corrupted party. A real execution of $\pi = (\pi_1, \pi_2)$ on inputs (x_1, x_2), auxiliary input[3] z to $\mathcal{A}$ and the security parameter λ, denoted by $\mathrm{REAL}^{\mathsf{cf\text{-}sfe}}_{\pi, \mathcal{A}(z), i}(x_1, x_2, \lambda)$ is defined as the output of the honest party and the adversary upon execution of π.

The ideal world. Let $i \in \{1, 2\}$ be the index of the corrupted party. We define the ideal world in two steps. During the ideal execution, the honest party sends its input x_{3-i}, and a uniformly random string r_{3-i} used by the commitment scheme, to the trusted party. Party P_i which is controlled by the ideal adversary $\mathcal{S}$, called the simulator, may either abort (sending a special symbol $\perp$) or send input x'_i, and an arbitrary randomness r'_i (not necessarily uniform) chosen based on the auxiliary input z, and P_i's original input x_i. Denote by $((x'_1, r'_1), (x'_2, r'_2))$ the values received by the trusted party. If the trusted party receives $\perp$, the value $\perp$ is forwarded to both P_1 and P_2 and the ideal execution terminates; else the trusted party computes $\gamma_1 = \mathsf{C}_1(x'_1; r'_1)$ and $\gamma_2 = \mathsf{C}_2(x'_2; r'_2)$, respectively. The TTP sends γ_{3-i} to $\mathcal{S}$, which can either continue or abort by sending $\perp$ to the TTP. In case of an abort, the TTP sends $\perp$ to the honest party; otherwise, it sends γ_i.

In the second phase, the honest party continues the ideal execution by sending to the TTP a continue flag, or aborts by sending $\perp$. $\mathcal{S}$ sends either $\perp$ or continue based on the auxiliary input z, P_i's original input, and the value γ_{3-i}. If the trusted party receives $\perp$, the value $\perp$ is forwarded to both P_1 and P_2 and the ideal execution terminates; else the trusted party computes $y_1 = f_1(x'_1, x'_2)$ (resp. $y_2 = f_2(x'_1, x'_2)$).

[3] As usual, the auxiliary input represents some a priori information the adversary might know about the parties' inputs to the protocol.

The TTP sends y_i to $\mathcal{S}$. At this point, $\mathcal{S}$ can decide whether the trusted party should continue, and thus send the output y_{3-i} to the honest party, or halt, in which case the honest party receives $\perp$. The honest party outputs the received value. The simulator $\mathcal{S}$ outputs an arbitrary polynomial-time computable function of $(z, x_i, y_i, \gamma_{3-i})$.

The ideal execution of f on inputs (x_1, x_2), auxiliary input z to $\mathcal{S}$ and security parameter λ, denoted by $\text{IDEAL}^{\mathsf{cf\text{-}sfe}}_{f,\mathsf{C}_1,\mathsf{C}_2,\mathcal{S}(z),i}(x_1, x_2, \lambda)$ is defined as the output of the honest party and the simulator.

Emulating the ideal world. We define a secure commit-first protocol π as follows:

Definition 3.1 (Commit-First Protocols)**.** *Let π and f be as above. We say that π is* a commit-first protocol (w.r.t. $\mathsf{C}_1, \mathsf{C}_2$) for computing $f = (f_1, f_2)$ in the presence of malicious adversaries with abort *if for every non-uniform PPT adversary $\mathcal{A}$ in the real world there exists a non-uniform PPT simulator $\mathcal{S}$ in the ideal world, such that for every $i \in \{1, 2\}$,*

$$\left\{\text{REAL}^{\mathsf{cf\text{-}sfe}}_{\pi,\mathcal{A}(z),i}(x_1, x_2, \lambda)\right\}_{x_1,x_2,z,\lambda} \equiv_c \left\{\text{IDEAL}^{\mathsf{cf\text{-}sfe}}_{f,\mathsf{C}_1,\mathsf{C}_2,\mathcal{S}(z),i}(x_1, x_2, \lambda)\right\}_{x_1,x_2,z,\lambda}$$

where $x_1, x_2, z \in \{0,1\}^*$, with $|x_1| = |x_2|$, and $\lambda \in \mathbb{N}$.

Notice that the above definition assumes that the parties (and adversary) know the input lengths. We remark that such an assumption is unavoidable, as to some extent some information on the inputs length can always be inferred (e.g., in the case of encryption).

3.2 Instantiations

3.2.1 The GMW Compiler

The GMW compiler [GMW87] is a general transformation for compiling any SFE with security against semi-honest adversaries into one with security against malicious ones. We observe that the malicious SFE resulting from the GMW compiler is also a commit-first protocol. This is of theoretical interest: *when combined with the compilers we introduce in this paper, it yields a general compiler for transforming any semi-honest SFE into a rate-limited malicious SFE.*

Recall the GMW compiler which consists of the following three phases.

(i) **Input-committing phase:** Each party commits to the input he will use in the protocol.

(ii) **Coin-generation phase:** Each party receives a uniformly random string to use in the protocol emulation phase, while its counterpart obtains a commitment to that randomness.

(iii) **Protocol emulation phase:** Parties engage in a protocol where each message of the semi-honest SFE is accompanied with proofs of correctness of the computation and its consistency with the committed inputs and randomness.

We refer the reader to [Gol09] for a complete description of the compiler and a proof that it yields an SFE with security against malicious adversaries. It turns out that the input-committing phase of the GMW compiler is exactly what we need in a commit-first protocol. In particular, the functionality of the input-committing phase for the first party is defined in [Gol09] as $((x, r), 1^\lambda) \to ((x, r), \mathsf{C}(x; r))$, where C is a hiding and binding commitment scheme. Then, a construction is provided that realizes this functionality in presence of malicious adversaries.[4] This is identical to the functionality computed by the trusted party (once for P_1 and once for P_2) in the first phase of the ideal execution for a commit-first SFE we defined above. This observation yields the following claim.

[4] Roughly speaking, the commitment is accompanied with a zero-knowledge proof of knowledge of the input and the randomness fed to the commitment.

Claim 3.2. *The GMW compiler transforms any SFE protocol with security against semi-honest adversaries into a commit-first SFE protocol with security against malicious adversaries.*

3.2.2 Yao's Garbled Circuits Protocol

Here, we investigate the commit-first property of Yao's garbled circuit protocol [Yao82, Yao86]. Yao's protocol is one of the most important general-purpose two-party SFE constructions. In particular, due to its desirable efficiency properties, it has been the subject of multiple software implementations [MNPS04, HKS+10, HEKM11]. Currently, the most efficient method for making Yao's protocol secure against malicious adversaries is the cut-and-choose approach [MF06, LP07].

We observe that Yao's protocol is *one-sided* commit-first. In other words, one of the parties in the protocol commits to his input during the protocol in such a way that the simulator in the ideal world is able to extract the corresponding message and randomness. We note that the one-sided commit-first property is indeed sufficient for many applications of rate-limited SFE where one is only interested in monitoring the rate for one party. For instance, this is the case in most client-server applications, where the server enforces the rate limit on the client.

Commit-first Yao via commit-first OT. We do not discuss the details of Yao's constructions here and refer the reader to [LP09] for a detailed description. However, we recall that in the cut-and-choose approach, the first party (garbler) computes multiple garbled circuits while the second party (evaluator) evaluates a fraction of these circuits. To proceed with the evaluation, the first step taken by the evaluator in the protocol is a *series of oblivious transfers* (OTs), one for each bit of its input. The evaluator plays the role of the receivers in each OT, and uses one input bit in each. The garbler plays the role of the sender, and uses two garbled values (corresponding to an input wire) as its input. For Yao's garbled circuit protocol to be one-sided commit-first (with respect to the evaluator in this case), we simply need to make sure that the oblivious transfer being used is commit-first itself (in addition to being secure against malicious adversaries). In particular, we need a guarantee that the receiver in the OT is committed to its input, and that the simulator in the security proof is able to extract both the message and the randomness used in the commitment.

Commit-first OT. There are multiple OT constructions that satisfy the commit-first property. For example, consider the general construction of [Lin08] based on any homomorphic encryption. We observe that if the homomorphic encryption scheme being used is randomness-recovering (i.e., the decryption algorithm recovers both the message and the randomness), the resulting OT will be commit-first. Consequently, the instantiation of their construction based on Paillier's encryption [Pai99] yields a commit-first OT.

A second option is to use the OPE construction we will discuss in Section 7 to instantiate the OT in the Yao's protocol. Note that the OPE problem is a generalization of oblivious transfer. The OT sender with an input pair (a_0, a_1) can let its polynomial be $p(x) = a_0(1-x)+a_1x$, while the receiver can use its input bit b as the input to the polynomial. It is easy to see that $p(b) = a_b$ for $b \in \{0, 1\}$. There is one small issue with this OT construction: It is not fully-secure against malicious adversaries in the form we describe it. In particular, a malicious receiver can choose a value for b that is not a bit. This issue can, however, be easily fixed by adding an efficient ZK proof of the statement that the plaintext corresponding to $\mathsf{E}(pk, b)$ is either the message 0 or 1. Since the OPE construction we discussed is commit-first, so is the resulting commit-first OT.

Summarizing the above discussion, we conclude with the following claim:

Claim 3.3. *When instantiated via a commit-first OT, cut-and-choose compilations of Yao's garbled circuit, are one-sided commit-first (with respect to the circuit evaluator) with security against malicious adversaries.*

3.2.3 Secure 2PC of Jarecki-Shmatikov

Jarecki and Shmatikov [JS07] design a variant of Yao's garbled circuits protocol for securely computing any two-party circuit on *committed inputs*. Their protocol is secure in a universally composable way in the presence of malicious adversaries, in the common reference string (CRS) model.

Their construction starts by having the two parties commit to their inputs. Then, a variant of Yao's protocol is design to operate on these committed inputs. Both the commitment scheme and the symmetric-key encryption needed in Yao's garbled circuit construction are instantiated via a simplified variant of the Camenisch-Shoup (CS) encryption scheme [CS03]. The computation is accompanied with efficient ZK proofs that are specially designed to work with the CS scheme. We refer the reader to [JS07] for a complete description of their construction.

Their construction can be easily transformed to a commit-first protocol in the CRS model. The protocol starts with each party committing to its input and proving the validity of the commitment. As mentioned above, the commitment scheme used is a simplified CS encryption. Unfortunately, knowing the secret key for the encryption scheme does not allow one to recover the randomness used for encryption as well. Our commit-first ideal execution, instead, requires this property. In other words, the simulator in the proof needs to send both the message and the randomness used by the commitment scheme to the TTP. However, as mentioned by the authors themselves, a wide range of other commitment schemes can also be used for this purpose. To satisfy the commit-first property, we simply need to make sure that the randomness used in the commitment is recoverable given a trapdoor (or the secret key itself). This is efficiently realizable, for example, using Paillier's encryption scheme [Pai99], in which the decryption algorithm recovers the randomness as well, hence yielding a commit-first variant of their construction.

Claim 3.4. *The two-party protocol of [JS07] is a commit-first SFE with security against malicious adversaries, in the CRS model.*

3.2.4 PSI Protocol of Hazay and Nissim

In the private set intersection (PSI) problem, two parties P_1 and P_2, hold the sets X and Y. Their goal is for one or both parties to learn $X \cap Y$ without revealing additional information about their sets. The PSI problem has been the focus of active research, and to date, many constructions with a variety of efficiency and security properties have been designed and implemented [FNP04, CKT10, HN12, HEK12].

Here, we focus on the protocol of Hazay and Nissim [HN12], since it is secure against malicious adversaries and we can easily show it to be a one-sided commit-first protocol as well. Once again, we do not describe the details of their construction but mostly focus on the components we need to prove the commit-first property. In particular, we observe that one of the parties engaged in the protocol, say P_1 holding the set $X = \{x_1, \ldots, x_n\}$ starts by computing the commitments $\mathsf{C}(x_i; r_i)$ for $1 \leq i \leq n$ and proving knowledge of x_i and r_i to P_2. This indeed constitutes a commitment to the set X, and allows the simulator in the proof to extract both the set X and the randomness used in the commitments, hence yielding a commit-first protocol with respect to P_1.

Claim 3.5. *The private set intersection protocol of [HN12] is one-sided commit-first with security against malicious adversaries.*

3.2.5 Oblivious Automata Evaluation of Gennaro, Hazay and Sorensen

In an oblivious automata evaluation (OAE) protocol, party P_1 holds a description of an automaton Γ whereas party P_2 holds a string x. After the execution of OAE, party P_1 obtains $\Gamma(x)$, and P_2 learns nothing.

In [GHS10], Gennaro, Hazay and Sorensen introduce a secure OAE protocol in presence of malicious adversaries. At a high level, party P_1 and P_2 first agree on a public key for an encryption scheme. Then, party P_1 sends the transition table of Γ in encrypted form together with a ZK-proof of its validity. Party P_2 is then able to work on this ciphertext in order to evaluate its input string x. Eventually, P_2 proves validity of the last ciphertext.

Similar to the OPE protocol in Section 7, the encryption and its proof of validity can be seen as a commitment by party P_1 to his input. This takes place before the actual automata evaluation is performed. However, party P_2 proves validity at the end of the execution. Hence, we observe here a one-sided commit-first oblivious automata evaluation protocol with respect to party P_1.

Claim 3.6. *The oblivious automata evaluation protocol of [GHS10] is a one-sided commit-first OAE with security against malicious adversaries.*

4 Rate-Limited SFE

In this section, we introduce three notions for rate-limited secure function evaluation (RL-SFE). In particular, we augment the standard notion of two-party SFE by allowing each player to monitor and/or limit, the number of distinct inputs (the *rate*) the other player uses in multiple executions. The idea is that each party can abort the protocol if the number of distinct inputs used in the previous executions raises above a threshold $\mathscr{n} \in \mathbb{N}$. We call this threshold the *rate limit*, i.e. the maximum number of allowable executions with distinct inputs.

Naturally, our security definitions for RL-SFE are concerned with *multiple* executions of an SFE protocol and reduce to the standard (stand-alone) simulation-based definition for SFE, under static corruptions, when applied to a single run. We call a sequence of executions of a protocol π $(\mathscr{n}_1, \mathscr{n}_2)$-limited if party P_1 (resp. P_2) can use at most $\mathscr{n}_1$ (resp. $\mathscr{n}_2$) distinct inputs in the executions. In this work, we assume that the executions take place *sequentially*, i.e. one execution after the other. We emphasize that the inputs used by the parties in each execution can depend on the transcripts of the previous executions, but honest parties will always use fresh randomness in their computation.

As discussed in Section 1.1, we provide three security definitions for rate-limited SFE: (i) rate-hiding, (ii) rate-revealing, and (iii) pattern-revealing. In a *rate-hiding* RL-SFE, at the end of each execution, the only information revealed to the parties (besides the output from the function being computed), is whether the agreed-upon rate limit (threshold) has been exceeded or not, but nothing else. In a *rate-revealing* RL-SFE, in addition to the above, parties also learn the current rates (i.e., the number of distinct inputs used by their counterpart so far). Finally, in a *pattern-revealing* RL-SFE, parties further learn the pattern of occurrences of each others' inputs in the previous executions. In particular, each party learns which executions were invoked by the same input and which ones used different ones, but nothing else.

High-level description. Let $f = (f_1, f_2)$ be a pair of polynomial-time functions such that f_i is of type $f_i : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$. Consider an arbitrary polynomial number ℓ of sequential executions of a two-party SFE protocol π for evaluating f on parties' inputs. During the j-th execution, party P_i has input x_i^j and should learn $y_i^j = f_i(x_1^j, x_2^j)$. We will define rate-limited SFE in the general case where *both* parties are allowed to change their input in each execution. The case of oracle attacks and secure metering, where one party's input is fixed and the other party's input changes, are found as a special case. (In the case of secure metering one can also think that a change in the service provider's input reflects a software update.)

In the ideal world, during the j-th execution, each party sends its input to a trusted authority. The following is then performed for both $i = 1, 2$. The trusted party checks whether value x_i^j was already sent in a previous execution; in case it was not, a new entry (x_i^j, j) is stored in an initially empty array $\mathcal{X}_i$. Else, the smallest index $j' < j$ corresponding to such input is recovered. Whenever $\#\mathcal{X}_i$ exceeds $\mathscr{r}_i$ the trusted party aborts. Otherwise, the current outputs $y_i^j = f_i(x_1^j, x_2^j)$ are computed. Finally: (i) in the rate-hiding definition party P_i learns only y_i^j; (ii) in the rate-revealing definition party P_i learns also $\#\mathcal{X}_{3-i}$, i.e. the (partial) total number of distinct inputs used by P_{3-i} until the j-th execution; (iii) in the pattern-revealing definition party P_i learns j', i.e. the index corresponding to the query where x_i^j was asked for the first time. Note that if the rate is exceeded, the trusted party aborts here, but, equivalently, we could simply ignore this execution and still allow to query previous inputs in subsequent executions.

We formalize the above intuitive security notions for all three flavors using the simulation-based ideal/real world paradigm. We first review the real execution which all three notions share.

The real world. In each execution, a non-uniform adversary $\mathcal{A}$ following an arbitrary efficient strategy can send messages in place of the corrupted party (whereas the honest party continues to follow π). Let $i \in \{1, 2\}$ be the index of the corrupted party. The j-th real execution of π on inputs (x_1^j, x_2^j), auxiliary input z^j to $\mathcal{A}$ and security parameter λ, denoted by $\textsc{real}^{\mathscr{r}}_{\pi,\mathcal{A}(z^j),i}(x_1^j, x_2^j, \lambda)_j$ is defined as the output of the honest party and the adversary in the j-th real execution of π. We denote by $\textsc{real}^{\mathscr{r}}_{\pi,\mathcal{A}(\mathbf{z}),i}(\mathbf{x}_1, \mathbf{x}_2, \lambda, \ell)$ the accumulative distribution at the end of the ℓ-th execution, i.e.,

$$\textsc{real}^{\mathscr{r}}_{\pi,\mathcal{A}(\mathbf{z}),i}(\mathbf{x}_1, \mathbf{x}_2, \lambda, \ell) = \textsc{real}^{\mathscr{r}}_{\pi,\mathcal{A}(z^1),i}(x_1^1, x_2^1, \lambda)_1, \ldots, \textsc{real}^{\mathscr{r}}_{\pi,\mathcal{A}(z^\ell),i}(x_1^\ell, x_2^\ell, \lambda)_\ell$$

where $\mathbf{x}_1 = (x_1^1, \ldots, x_1^\ell)$, $\mathbf{x}_2 = (x_2^1, \ldots, x_2^\ell)$ and $\mathbf{z} = (z^1, \ldots, z^\ell)$.

The ideal world. The trusted party keeps two arrays $\mathcal{X}_1$, and $\mathcal{X}_2$ initially set to $\emptyset$. Let $i \in \{1, 2\}$ be the index of the corrupted party. During the j-th ideal execution, the honest party sends its input to the trusted party. Party P_i, which is controlled by the ideal adversary $\mathcal{S}$, called the simulator, may either abort (sending a special symbol $\perp$) or send input $x_i'^j$ to the trusted party chosen based on the auxiliary input z^j, P_i's original input x_i^j, and its view in the previous $j - 1$ ideal executions. Denote with $(x_1'^j, x_2'^j)$ the values received by the trusted party (note that if $i = 2$ then $x_1'^j = x_1^j$).

If the trusted party receives $\perp$, the value $\perp$ is forwarded to both P_1 and P_2 and the ideal execution terminates; else when the trusted party receives $x_1'^j$ as the first party's input, it checks whether an entry $(x_1'^j, j') \in \mathcal{X}_1$ already exists; if so, it sets $J_1 = j'$ for the smallest such j'. Otherwise, it creates a new entry $(x_1'^j, j)$, adds it to $\mathcal{X}_1$, and sets $J_1 = j$. An identical procedure is applied to the input of the second party $x_2'^j$ to determine an index J_2. At the end of the j-th ideal execution if $\sigma_1 := \#\mathcal{X}_1 \geq \mathscr{r}_1$ *or* $\sigma_2 := \#\mathcal{X}_2 > \mathscr{r}_2$, the value $\perp$ is forwarded to both P_1 and P_2 and the ideal execution terminates. Otherwise, the pair $(y_1^j, y_2^j) = (f_1(x_1'^j, x_2'^j), f_2(x_1'^j, x_2'^j))$ is computed.

At this point, the ideal output will be different depending on the variant of RL-SFE being considered.

Rate-Hiding. The trusted party forwards to the malicious party P_i the output y_i^j. At this point, $\mathcal{S}$ can decide whether the trusted party should continue, and thus send the pair y_{3-i} to the honest party, or halt, in which case the honest party receives $\perp$.

Rate-Revealing. The trusted party forwards to the malicious party P_i the pair (y_i^j, σ_{3-i}). At this point, $\mathcal{S}$ can decide whether the trusted party should continue, and thus send the pair (y_{3-i}^j, σ_i) to the honest party, or halt, in which case the honest party receives $\perp$.

Pattern-Revealing. The trusted party forwards to the malicious party P_i the pair (y_i^j, J_{3-i}). The integer $1 \leq J_{3-i} \leq j$ represents the index of the first execution where the input x_{3-i}^j has been used. At this point, $\mathcal{S}$ can decide whether the trusted party should continue, and thus send the pair (y_{3-i}^j, J_i) to the honest party, or halt, in which case the honest party receives $\perp$.

The honest party outputs the received value. The simulator $\mathcal{S}$ outputs an arbitrary polynomial-time computable function of (z^j, x_i^j, y_i^j).

The j-th ideal execution of f on inputs (x_1^j, x_2^j), auxiliary input z^j to $\mathcal{S}$ and security parameter λ, denoted by $\textsc{ideal}^{\mathscr{r}-\mathsf{type}}_{f,\mathcal{S}(z^j),i}(x_1^j, x_2^j, \lambda)_j$ is defined as the output of the honest party and the simulator in the above j-th ideal execution. Here, $\mathsf{type} \in \{\mathsf{rh}, \mathsf{rr}, \mathsf{pr}\}$ determines the flavor of rate-limited SFE. We denote by $\textsc{ideal}^{\mathscr{r}-\mathsf{type}}_{f,\mathcal{S}(\mathbf{z}),i}(\mathbf{x}_1, \mathbf{x}_2, \lambda, \ell)$ the accumulative distribution at the end of the ℓ-th execution, i.e.,

$$\textsc{ideal}^{\mathscr{r}-\mathsf{type}}_{f,\mathcal{S}(\mathbf{z}),i}(\mathbf{x}_1, \mathbf{x}_2, \lambda, \ell) = \textsc{ideal}^{\mathscr{r}-\mathsf{type}}_{f,\mathcal{S}(z^1),i}(x_1^1, x_2^1, \lambda)_1, \ldots, \textsc{ideal}^{\mathscr{r}-\mathsf{type}}_{f,\mathcal{S}(z^\ell),i}(x_1^\ell, x_2^\ell, \lambda)_\ell$$

where $\mathbf{x}_1 = (x_1^1, \ldots, x_1^\ell)$, $\mathbf{x}_2 = (x_2^1, \ldots, x_2^\ell)$ and $\mathbf{z} = (z^1, \ldots, z^\ell)$.

Emulating the ideal world. Roughly speaking, ℓ sequential executions of a protocol π are secure under the rate limit $\mathscr{r} = (\mathscr{r}_1, \mathscr{r}_2)$ if the real executions can be simulated in the above mentioned ideal world. More formally, we define a secure $(\mathscr{r}_1, \mathscr{r}_2)$-limited protocol π as follows:

Definition 4.1 (RL-SFE). *Let π and f be as above, and consider $\ell = poly(\lambda)$ sequential executions of protocol π. For* $\mathsf{type} \in \{\mathsf{rh}, \mathsf{rr}, \mathsf{pr}\}$*, we say protocol π is a* secure type $\mathscr{r}$-limited SFE *for computing $f = (f_1, f_2)$,* in presence of malicious adversaries with abort *with $\mathscr{r} = (\mathscr{r}_1, \mathscr{r}_2)$, if for every non-uniform PPT adversary $\mathcal{A}$ there exists a non-uniform PPT simulator $\mathcal{S}$, such that for every $i \in \{1, 2\}$,*

$$\left\{\textsc{real}^{\mathscr{r}}_{\pi,\mathcal{A}(\mathbf{z}),i}(\mathbf{x}_1, \mathbf{x}_2, \lambda, \ell)\right\}_{\mathbf{x}_1,\mathbf{x}_2,\mathbf{z},\lambda} \equiv_c \left\{\textsc{ideal}^{\mathscr{r}-\mathsf{type}}_{f,\mathcal{S}(\mathbf{z}),i}(\mathbf{x}_1, \mathbf{x}_2, \lambda, \ell)\right\}_{\mathbf{x}_1,\mathbf{x}_2,\mathbf{z},\lambda}$$

where $\mathbf{x}_1, \mathbf{x}_2, \mathbf{z} \in (\{0,1\}^*)^\ell$, with $|\mathbf{x}_1[j]| = |\mathbf{x}_2[j]|$ for all j, and $\lambda \in \mathbb{N}$.

We remark that, while we only consider static corruptions, the corrupted player is allowed to choose its input in each of the sequential executions in a fully adaptive manner (based on the view so far). Also note that, in applications, the parameter $\mathscr{r}$ typically depends on the function f being evaluated; e.g., in the case of OPE $\mathscr{r}$ is upper bounded by the degree of the polynomial. The value ℓ, instead, is a parameter of the security definition, but we decided to leave it implicit as our compilers produce protocols which are secure for any polynomial number of sequential repetitions.

5 Compilers for Rate-Limited SFE

In this section, we introduce our three compilers to transform an arbitrary (two-party) CF-SFE protocol into a *rate-limited* protocol for the same functionality.

Our first compiler Ψ_{rh} achieves the notion of rate-hiding RL-SFE through the use of ZK arguments and public-key encryption. Our second compiler Ψ_{rr} achieves the notion of rate-revealing RL-SFE and is more efficient in that it needs to prove a simpler statement and does not rely on encryption. Our last compiler Ψ_{pr} introduces essentially no overhead and avoids the use of zero-knowledge, yielding our third notion of pattern-revealing RL-SFE.

Let π_f be a two-party (single-run) commit-first protocol for secure function evaluation of a function $f = (f_1, f_2)$ (cf. Definition 3.1). Our compilers get as input (a description of) π_f, together with the rate $\mathpzc{r} = (\mathpzc{r}_1, \mathpzc{r}_2)$, and the number of executions ℓ, and output (a description of) $\widehat{\pi}_f \leftarrow \Psi(\pi_f, \mathpzc{r}, \ell)$. The compilers are functionality preserving, meaning that protocol $\widehat{\pi}_f$ repeatedly computes the same functionality f.

5.1 A Rate-Hiding Compiler

The overview. We naturally divide the CF-SFE protocol into a committing phase and a function evaluation phase and introduce a new phase in between where P_1 and P_2 convince each other that they have not exceeded the rate limit. The latter step is achieved as follows. Whenever one of the parties is going to use a "fresh" input, it transmits an encryption of "1" to the other party; otherwise, it sends an encryption of "0". The encryptions are obtained using a CPA-secure PKE scheme $(\mathsf{G}, \mathsf{E}, \mathsf{D})$. Then, the party proves in ZK that "the last commitment transmitted hides an already used input *and* it encrypted 0, *or* the last commitment transmitted hides a fresh input *and* it encrypted 1 *and* the sum of all the plaintexts, encrypted until now, does not exceed the rate". A successful verification of this proof convinces the other party that the rate is not exceeded, leaking nothing more than this. We instantiate such ZK proofs for the OPE problem in Section 7. Notice that to generate such a proof each party needs to store all the ciphertexts transmitted to the other player, together with all the inputs and randomness used to generate the previous commitments. On the other hand, to verify the other party's proof, one needs to store the ciphertexts and the commitments received in all earlier executions. The remainder of the messages exchanged during each execution, however, can be discarded.

The construction of our rate-hiding $\mathpzc{r}$-limited compiler Ψ_{rh} is depicted in Fig. 1.

Theorem 5.1. *Let $\pi_f = (\pi_f^1, \pi_f^2)$ be a commit-first protocol (w.r.t. $\mathsf{C}_1, \mathsf{C}_2$) securely evaluating function $f = (f_1, f_2)$, and assume that $\mathsf{C}_1, \mathsf{C}_2$ are perfectly binding and computationally hiding commitment schemes, that $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ is a CPA-secure PKE scheme, and that $(\mathcal{P}_i, \mathcal{V}_i)$ is a ZK argument system for the language L_i. Then $\widehat{\pi}_f \leftarrow \Psi_{\mathsf{rh}}(\pi_f, \mathpzc{r}_1, \mathpzc{r}_2, \ell)$ of Fig. 1 is a secure* rate-hiding *$(\mathpzc{r}_1, \mathpzc{r}_2)$-limited protocol for f.*

Proof. Consider an adversary $\mathcal{A} = (\mathcal{A}^1, \ldots, \mathcal{A}^\ell)$ corrupting party P_i during the ℓ executions of $\widehat{\pi}_f$. In particular, $\mathcal{A}^j$ represents $\mathcal{A}$'s strategy during the jth execution, and $\text{REAL}^{\mathpzc{r}}_{\widehat{\pi}_f,\mathcal{A}(z^j),i}(x_1^j, x_2^j, \lambda)_j$ denotes the distribution of its output. We denote by $\text{REAL}^{\mathpzc{r}\text{-}\mathsf{rh}}_{\widehat{\pi}_f,\mathcal{A}(\mathbf{z}),i}(\mathbf{x}_1, \mathbf{x}_2, \lambda, \ell)$ the joint distribution of the output of all the $\mathcal{A}^j$s combined. Note that each $\mathcal{A}^j$ passes the necessary state information (i.e., her view) to $\mathcal{A}^{j+1}$.

We describe a simulator $\mathcal{S} = (\mathcal{S}^1, \ldots, \mathcal{S}^\ell)$ in the ideal world—as discussed in Section 4—that mimics $\mathcal{A}$'s output. Before doing so, note that we are given as input to the compiler Ψ_{rh} (besides the rates and ℓ) the commit-first SFE protocol π_f. According to the security definition (cf. Definition 3.1), for any admissible adversary against π_f, there exists a simulator $\mathcal{S}_{\mathsf{cf}}$ that mimics her behavior in the CF-SFE's ideal world. Moreover, due to the way the CF-SFE ideal

Rate-Hiding Compiler Ψ_{rh}:

Let $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ be a public-key encryption scheme with message space $\{0,1\}$ and ciphertext space $\mathcal{E}$. Parties P_1 and P_2, respectively, hold an auxiliary key pair $(pk_i, sk_i) \leftarrow \mathsf{G}(1^\lambda)$. Given as input a commit-first protocol (w.r.t. $\mathsf{C}_1, \mathsf{C}_2$) $\pi_f = (\pi_f^1, \pi_f^2)$, a rate $\mathscr{r} = (\mathscr{r}_1, \mathscr{r}_2)$, and a number of executions ℓ, the compiled protocol $\widehat{\pi}_f$ is made of three phases, described below. Party P_1 and P_2 keep the state variables $\Sigma_i := (\Gamma_i, (\Omega_1, \Omega_2), \Lambda_i)$ initially set to be empty. For each execution $j \in [\ell]$, $\widehat{\pi}_f$ proceeds as follows:

Committing Phase: Parties P_1 and P_2, holding respectively inputs x_1^j and x_2^j, run the protocol π_f^1 yielding the output $((x_1^j, r_1^j, \gamma_2^j = \mathsf{C}_2(x_2^j; r_2^j)), (x_2^j, r_2^j, \gamma_1^j = \mathsf{C}_1(x_1^j; r_1^j)))$.

Proof of Repeated-Input Phase: When the input x_i^j of party P_i is *not* fresh—i.e., it has already been used in a previous execution—P_i computes $c_i^j \leftarrow \mathsf{E}(pk_i, 0)$. Otherwise, P_i computes $c_i^j \leftarrow \mathsf{E}(pk_i, 1)$ and lets $\Lambda_i := \Lambda_i \cup \{(x_i^j, r_i^j)\}$. Then, add also c_i^j to the state, i.e., $\Omega_i := \Omega_i \cup \{c_i^j\}$. Consider the following languages:

$$L_i^{\mathrm{rate}} = \left\{\Omega_i \subset \mathcal{E}^j :\ \textstyle\sum_{c \in \Omega_i} \mathsf{D}(sk_i, c) \leq \mathscr{r}_i\right\} \qquad L_i^b = \{c \in \mathcal{E} :\ \exists r \text{ s.t. } c = \mathsf{E}(pk_i, b; r)\}$$

$$L_i^{\mathrm{old}} = \left\{\gamma \in \mathcal{C}_i :\ \exists (x, r, r') \text{ s.t. } \gamma = \mathsf{C}_i(x; r) \text{ and } \mathsf{C}_i(x; r') \in \Gamma_{3-i}\right\},$$

and let $(\mathcal{P}_i, \mathcal{V}_i)$ be a ZK argument system for $L_i := (L_i^{\mathrm{old}} \wedge L^0) \vee (\overline{L_i^{\mathrm{old}}} \wedge L^1 \wedge L_i^{\mathrm{rate}})$. If $\#\Lambda_i \leq \mathscr{r}_i$, party P_i sends c_i^j and plays the role of the prover in $(\mathcal{P}_i, \mathcal{V}_i)$; otherwise, it outputs $\perp$ and aborts. Also, party P_i receives c_{3-i}^j from P_{3-i}, updates the state as in $\Omega_{3-i} := \Omega_{3-i} \cup \{c_{3-i}^j\}$ and plays the role of the verifier in $(\mathcal{P}_{3-i}, \mathcal{V}_{3-i})$. If the verification fails, it outputs $\perp$ and aborts. Otherwise, it lets $\Gamma_i := \Gamma_i \cup \{\gamma_{3-i}^j\}$ and proceeds to the next step.

Protocol Emulation Phase: P_1 and P_2 run the protocol π_f^2 on the same inputs as in the committing phase, yielding the output (y_1^j, y_2^j).

Figure 1: A compiler for rate-hiding rate-limited SFE.

world is defined, $\mathcal{S}_{\mathsf{cf}}$ can be naturally written as $\mathcal{S}_{\mathsf{cf}} = (\mathcal{S}_{\mathsf{cf}}^1, \mathcal{S}_{\mathsf{cf}}^2)$ where basically $\mathcal{S}_{\mathsf{cf}}^1$ emulates the commit-first phase (i.e., π_f^1), and passes its view to $\mathcal{S}_{\mathsf{cf}}^2$ who emulates the function evaluation phase (i.e., π_f^2).

The simulator $\mathcal{S} = (\mathcal{S}^1, \ldots, \mathcal{S}^\ell)$ picks $(pk, sk) \leftarrow \mathsf{G}(1^\lambda)$, runs a copy of $\mathcal{A}$, and keeps a state $\Sigma = (\Gamma, (\Omega_1, \Omega_2), \Lambda)$ initially set to be empty. The j-th execution is given below.

1. $\mathcal{S}^j$ takes $(x_i^j, \Sigma, pk, sk, z^j)$ as input.

2. In the committing phase, $\mathcal{S}^j$ invokes $\mathcal{S}_{\mathsf{cf}}^1$ on input x_i^j. The simulator $\mathcal{S}_{\mathsf{cf}}^1$ invokes $\mathcal{A}^j$ who controls party P_i in π_f^1. If $\mathcal{S}_{\mathsf{cf}}^1$ sends $\perp$ to its CF-SFE TTP, $\mathcal{S}^j$ sends $\perp$ to its own trusted party leading to an abort of the execution. Otherwise, $\mathcal{S}^j$ receives $x_i'^j, r_i'^j$ from $\mathcal{S}_{\mathsf{cf}}^1$ and computes $\gamma_i'^j = \mathsf{C}_i(x_i'^j; r_i'^j)$. It also samples a random $x_{3-i}'^j \in \mathcal{M}_{3-i}$, computes $\gamma_{3-i}'^j = \mathsf{C}_{3-i}(x_{3-i}'^j; r_{3-i}'^j)$ using uniform randomness $r_{3-i}'^j \in \mathcal{R}_{3-i}$ and sends the result to $\mathcal{A}^j$. If $(x_{3-i}'^j, *) \notin \Lambda$, update $\Lambda := \Lambda \cup \{(x_{3-i}'^j, r_{3-i}'^j)\}$.

3. $\mathcal{S}^j$ sends $x_i'^j$ to its TTP, and receives $y_i^j = f_i(x_1'^j, x_2'^j)$ back, where $x_{3-i}'^j = x_{3-i}^j$. Recall that $y_i^j = \perp$ shows whether one of the parties has exceeded its own rate. If the returned value is $y_i^j = \perp$, the simulator sends $\perp$ to $\mathcal{A}^j$ and terminates the execution. On the other hand, if the returned value is $y_i^j \neq \perp$ and $x_{3-i}'^j$ has never been used before, $\mathcal{S}^j$ computes $c_{3-i}'^j \leftarrow \mathsf{E}(pk, 1)$. Otherwise, it computes $c_{3-i}'^j \leftarrow \mathsf{E}(pk, 0)$ and stores the generated ciphertext, i.e., $\Omega_{3-i} := \Omega_{3-i} \cup \{c_{3-i}'^j\}$.

   Hence, the simulator forwards $c_{3-i}'^j$ to $\mathcal{A}^j$ and runs internally the ZK simulator $\mathcal{S}_{\mathsf{zk}}$ for the argument system with language

$$(L^{\text{old}}_{3-i} \wedge L^{0}_{3-i}) \vee (\overline{L^{\text{old}}_{3-i}} \wedge L^{1}_{3-i} \wedge L^{\text{rate}}_{3-i}),$$

playing the role of the prover (with $\mathcal{A}^j$ being the verifier). (Note that the last step involves the state Σ). Notice that $\mathcal{S}_{\mathsf{zk}}$ may itself need to rewind $\mathcal{A}^j$. However, this is not an issue because our simulator $\mathcal{S}$ invokes different simulators sequentially; in particular, $\mathcal{A}^j$ will be in a consistent state when $\mathcal{S}_{\mathsf{zk}}$ terminates and thus $\mathcal{S}$ can proceed with the rest of the simulation.

$\mathcal{S}^j$ also receives c'^j_i from $\mathcal{A}^j$, plays the role of the verifier in the argument system (with $\mathcal{A}^j$ being the prover) and updates the state to $\Omega_i := \Omega_i \cup \{c'^j_i\}$ and $\Gamma = \Gamma \cup \{\gamma'^j_i\}$.

4. Finally, $\mathcal{S}^j$ invokes $\mathcal{S}^2_{\mathsf{cf}}$ on input $(x'^j_i, r'^j_i, \gamma'^j_{3-i})$; $\mathcal{S}^2_{\mathsf{cf}}$ itself runs $\mathcal{A}^j$ who controls party P_i in π^2_f. We emphasize $\mathcal{S}^2_{\mathsf{cf}}$ does not run a new instance of $\mathcal{A}^j$ but it continues with running the same instance that has been running so far. If $\mathcal{S}^2_{\mathsf{cf}}$ sends $\perp$, $\mathcal{S}^j$ sends $\perp$ to its trusted party leading to an abort of the execution. Else, $\mathcal{S}^2_{\mathsf{cf}}$ sends the $\mathsf{continue}$ flag. $\mathcal{S}^j$ replies (on behalf of the CF-SFE TTP) by sending to $\mathcal{S}^2_{\mathsf{cf}}$, the output y^j_i he obtained earlier in the simulation.

5. Afterwards, $\mathcal{S}^j$ passes y^j_i to $\mathcal{A}^j$ and outputs whatever $\mathcal{A}^j$ does.

We now need to show that

$$\text{IDEAL}^{\mathscr{r}\text{-}\mathsf{rh}}_{f,\mathcal{S}(\mathbf{z}),i}(\mathbf{x}_i, \mathbf{x}_{3-i}, \lambda, \ell) \equiv_c \text{REAL}^{\mathscr{r}}_{\widehat{\pi}_f,\mathcal{A}(\mathbf{z}),i}(\mathbf{x}_i, \mathbf{x}_{3-i}, \lambda, \ell).$$

By a standard hybrid argument (losing a factor $1/\ell$ in the computational distance), it suffices to show indistinguishability for a single execution (i.e., the jth execution). Therefore, we need to show that for all $i \in \{1, 2\}$ and $j \in [\ell]$

$$\text{IDEAL}^{\mathscr{r}\text{-}\mathsf{rh}}_{f,\mathcal{S}(z^j),i}(x'^j_i, x'^j_{3-i}, \lambda)_j \equiv_c \text{REAL}^{\mathscr{r}}_{\widehat{\pi}_f,\mathcal{A}(z^j),i}(x^j_i, x'^j_{3-i}, \lambda)_j.$$

We consider a series of intermediate hybrid experiments. In the first experiment, we modify the simulator by letting it abort the execution on the basis of the verification of the ZK proofs, as it would be done in a real execution of the protocol. We argue that this modification is not distinguishable by the adversary $\mathcal{A}^j$ due to the soundness of the ZK argument system. In the second experiment, we assume that in contrast to the simulation above, the real input of the honest party is used in the simulation. We argue that this modification is not distinguishable by the adversary $\mathcal{A}^j$ due to the hiding property of the commitment and the IND-CPA security of the PKE scheme. In the last experiment, we replace the simulated ZK argument with a real one. The indistinguishability of the last two experiments follows naturally from the zero-knowledge property of the argument system. Finally, it is easy to see that the distribution of $\mathcal{A}^j$'s output in the last experiment is identical to the distribution of its output in the real protocol, which concludes our proof. Details follow.

Hybrid $\textbf{H}\textsc{yb}^1_{\mathcal{A}(z^j)}(x'^j_i, x'^j_{3-i}, \lambda)_j$: In the first hybrid experiment, we replace $\mathcal{S}^j$ by $\mathcal{S}^j_1$ who controls P_i in the ideal world. Essentially $\mathcal{S}^j_1$ is different from $\mathcal{S}^j$ merely in the way it aborts the simulation based on the execution of the argument system. Namely, in case the rate $\mathscr{r}_{3-i}$ is not exceeded, instead of looking at the output from the trusted party (cf. item 3. in the description of $\mathcal{S}^j$), it first plays the role of the verifier as party P_{3-i} would do in a real execution of the protocol. Hence, if the verification fails the value $\perp$ is sent to $\mathcal{A}^j$ and the execution is halted. Everything else is identical to the previous simulation. Next, we argue that

$$\text{IDEAL}^{\mathcal{R}\text{-rh}}_{f,\mathcal{S}(z^j),i}(x'^j_i, x'^j_{3-i}, \lambda)_j \equiv_c \text{HYB}^1_{f,\mathcal{A}(z^j),i}(x'^j_i, x'^j_{3-i}, \lambda)_j.$$

In fact, the modification above only affects the way the execution is halted. Denote with bad the event that $\mathcal{A}$ is able to come up with an accepting proof for a false statement, i.e., $\mathcal{A}$ is able to convince P_{3-i} that the rate $\mathcal{R}_i$ is not exceeded even though it already reached the rate itself. Note that the distribution produced by the two experiments above is identical provided that bad does *not* happen. Due to the soundness property of the ZK argument system, we must conclude that bad happens at most with negligible probability, thus showing that the two experiments are computationally indistinguishable.

Hybrid $\textbf{H}\textbf{YB}^2_{\mathcal{A}(z^j)}(x'^j_i, x^j_{3-i}, \lambda)_j$: In the second hybrid experiment, we replace $\mathcal{S}^j_1$ by $\mathcal{S}^j_2$ who controls P_i in the ideal world and at the same time plays the role of the TTP playing all the roles by itself. As a result, the simulator directly interacts with P_{3-i} during the ideal execution of the commit-first protocol. Essentially, $\mathcal{S}^j_2$ is identical to $\mathcal{S}^j_1$ with the exception that it is able to compute and send the correct commitment $\gamma^j_{3-i} = \mathsf{C}_{3-i}(x^j_{3-i}; r^j_{3-i})$ to $\mathcal{A}^j$. Also, $\mathcal{S}^j_2$ is able to compute the correct ciphertext c^j_{3-i} on the basis of the “freshness” of the real input x^j_{3-i}. Everything else is analogous to the previous simulation.

Next, we argue that

$$\text{HYB}^1_{f,\mathcal{A}(z^j),i}(x'^j_i, x'^j_{3-i}, \lambda)_j \equiv_c \text{HYB}^2_{f,\mathcal{A}(z^j),i}(x'^j_i, x^j_{3-i}, \lambda)_j.$$

We first argue that the simulation of the values y^j_1, y^j_2 is perfect. This follows by the perfect binding property of C_i, which implies that the function f is evaluated on the very same input extracted by the simulator. Hence, the only difference between the previous hybrid and the hybrid world described above is that the real input of the honest party is used in the latter. In particular, $\mathcal{S}^j_2$ feeds both $\mathcal{S}^1_{\mathsf{cf}}$ and $\mathcal{S}^2_{\mathsf{cf}}$ with the commitment $\mathsf{C}_{3-i}(x^j_{3-i}; r^j_{3-i})$ to the real input x^j_{3-i} as opposed to an arbitrary input x'^j_{3-i}; analogously the bit encrypted in c^j_{3-i} is chosen accordingly to x^j_{3-i}.

However, due to the hiding property of the commitment and the CPA-security of the PKE scheme, these two views are computationally indistinguishable. (In particular any distinguisher between the two experiments can be turned into an adversary breaking either the hiding property of the commitment or the CPA-security of the PKE scheme.) We rely here on the fact that, in both worlds, the simulator emulates the ZK proof using $\mathcal{S}_{\mathsf{zk}}$ as opposed to executing the real proof. In particular, the ZK simulator does not need the parties' private inputs for its simulation, and hence is not affected by the aforementioned change in the inputs.

Hybrid $\textbf{H}\textbf{YB}^3_{\mathcal{A}(z^j)}(x'^j_i, x^j_{3-i}, \lambda)_j$: We modify the previous hybrid world, by having $\mathcal{S}^j_3$ provide an actual ZK proof that the rate is not exceeded or to output $\perp$ if this is not case. For this purpose, $\mathcal{S}^j_3$ uses the state Σ and the current inputs x^j_1, x^j_2. The zero-knowledge property of the proof system automatically guarantees that the view generated using the real proof and the simulator $\mathcal{S}_{\mathsf{zk}}$ are computationally indistinguishable which in turn implies the computational indistinguishability of the output of the current hybrid experiment and the previous one. Thus, we have

$$\text{HYB}^2_{f,\mathcal{A}(z^j),i}(x'^j_i, x^j_{3-i}, \lambda)_j \equiv_c \text{HYB}^3_{f,\mathcal{A}(z^j),i}(x'^j_i, x^j_{3-i}, \lambda)_j.$$

To conclude the proof, it suffices to note that $\textsc{Hyb}^3_{f,\mathcal{A}(z^j),i}(x'^j_i, x^j_{3-i}, \lambda)_j$ exactly equals the output distribution of $\mathcal{A}^j$ in the real world, thus proving that $\widehat{\pi}_f \leftarrow \Psi_{\mathsf{rh}}(\pi_f, \mathscr{r}_1, \mathscr{r}_2, \ell)$ is a secure $\mathsf{rate\text{-}hiding}$ $(\mathscr{r}_1, \mathscr{r}_2)$-limited protocol for function f. □

5.2 A Rate-Revealing Compiler

The overview. Once again, we divide the CF-SFE protocol into a committing phase and a function evaluation phase and introduce a new phase in between where P_1 and P_2 convince each other that the current input has already been used in a previous execution. Note that the parties need to maintain a state variable Γ collecting the input commitments sent and received in all earlier executions. During the j-th execution, given a list of input commitments (and the corresponding inputs and randomness) for all the previous executions, party P_i can prove in ZK that the input commitment generated in the current execution is for the same value as one of the commitments collected previously; in case the input is repeated, party P_i simply omits the ZK proof. Party P_{3-i} also needs to collect the same set of commitments in order to verify the statement proven by P_i.

A complete description of the compiler is depicted in Fig. 2. We prove the following result:

Theorem 5.2. *Let $\pi_f = (\pi^1_f, \pi^2_f)$ be a commit-first protocol (w.r.t. $\mathsf{C}_1, \mathsf{C}_2$) securely evaluating function $f = (f_1, f_2)$, and assume that $\mathsf{C}_1, \mathsf{C}_2$ are perfectly binding and computationally hiding commitment schemes and that $(\mathcal{P}_i, \mathcal{V}_i)$ is a ZK argument system for the language L_i. Then $\widehat{\pi}_f \leftarrow \Psi_{\mathsf{rr}}(\pi_f, \mathscr{r}_1, \mathscr{r}_2, \ell)$ of Fig. 2 is a secure* $\mathsf{rate\text{-}revealing}$ *$(\mathscr{r}_1, \mathscr{r}_2)$-limited protocol for f.*

Proof. Consider an adversary $\mathcal{A} = (\mathcal{A}^1, \ldots, \mathcal{A}^\ell)$ corrupting party P_i during the ℓ executions of $\widehat{\pi}_f$. In particular, $\mathcal{A}^j$ represents $\mathcal{A}$'s strategy during the j-th execution, and its output distribution is denoted by $\textsc{real}^{\mathscr{r}}_{\widehat{\pi}_f,\mathcal{A}(z^j),i}(x^j_1, x^j_2, \lambda)_j$. We denote by $\textsc{real}^{\mathscr{r}}_{\widehat{\pi}_f,\mathcal{A}(\mathbf{z}),i}(\mathbf{x}_1, \mathbf{x}_2, \lambda, \ell)$ the joint distribution of the output of all the $\mathcal{A}^j$s combined. Note that each $\mathcal{A}^j$ passes the necessary state information (i.e., her view) to $\mathcal{A}^{j+1}$.

Rate-Revealing Compiler Ψ_{rr}:

Given as input a commit-first protocol (w.r.t. $\mathsf{C}_1, \mathsf{C}_2$) $\pi_f = (\pi^1_f, \pi^2_f)$, a rate $\mathscr{r} = (\mathscr{r}_1, \mathscr{r}_2)$, and a number of executions ℓ, the compiled protocol $\widehat{\pi}_f$ is made of three phases, described below. Party P_1 and P_2 keep the state variables $\Gamma_1, \Gamma_2 := \emptyset$, respectively. For each execution $j \in [\ell]$, $\widehat{\pi}_f$ proceeds as follows.

Committing Phase: Parties P_1 and P_2, holding respectively inputs x^j_1 and x^j_2, run the protocol π^1_f yielding the output $((x^j_1, r^j_1, \gamma^j_2 = \mathsf{C}_2(x^j_2; r^j_2)), (x^j_2, r^j_2, \gamma^j_1 = \mathsf{C}_1(x^j_1; r^j_1)))$.

Proof of Repeated-Input Phase: Consider the following language:

$$L_i = \{\gamma \in \mathcal{C}_i : \exists (x, r, r') \text{ s.t. } \gamma = \mathsf{C}_i(x; r) \wedge \mathsf{C}_i(x; r') \in \Gamma_{3-i}\},$$

and let $(\mathcal{P}_i, \mathcal{V}_i)$ be a ZK argument system for L_i. The following is executed for all $i \in \{1, 2\}$. When the input x^j_i of party P_i is *not* fresh—i.e., it has already been used in a previous execution—P_i plays the role of the prover in $(\mathcal{P}_i, \mathcal{V}_i)$. When the input x^j_i is fresh, P_i just forwards the empty string ε. Also, party P_i plays the role of the verifier in $(\mathcal{P}_{3-i}, \mathcal{V}_{3-i})$ (with P_{3-i} being the prover and L_{3-i} being the underlying language). If the value ε is received or if the verification of the proof fails, P_i updates the rate by letting $\mathscr{r}_{3-i} := \mathscr{r}_{3-i} - 1$ and the state by letting $\Gamma_i := \Gamma_i \cup \{\gamma^j_{3-i}\}$. If $\mathscr{r}_{3-i} < 0$, then party P_i output $\perp$ and aborts. Otherwise, if the verification is successful, the state and rate information will not be modified.

Protocol Emulation Phase: P_1 and P_2 run the protocol π^2_f on the same inputs as in the committing phase, yielding the output (y^j_1, y^j_2).

Figure 2: A compiler for rate-revealing rate-limited SFE.

We describe a simulator $\mathcal{S} = (\mathcal{S}^1, \ldots, \mathcal{S}^\ell)$ in the ideal world that mimics $\mathcal{A}$'s output. Our simulator makes use of the simulator $\mathcal{S}_{\mathsf{cf}} = (\mathcal{S}_{\mathsf{cf}}^1, \mathcal{S}_{\mathsf{cf}}^2)$ which exists due to the fact that our compiler takes as input a CF-SFE protocol π_f. The simulator $\mathcal{S} = (\mathcal{S}^1, \ldots, \mathcal{S}^\ell)$ runs a copy of $\mathcal{A}$, and keeps a state Γ_i initially set to be empty. The j-th execution is given below.

1. $\mathcal{S}^j$ takes (x_i^j, Γ_i, z^j) as input.

2. In the committing phase, $\mathcal{S}^j$ invokes $\mathcal{S}_{\mathsf{cf}}^1$ on input x_i^j. The simulator $\mathcal{S}_{\mathsf{cf}}^1$ invokes $\mathcal{A}^j$ who controls party P_i in π_f^1. If $\mathcal{S}_{\mathsf{cf}}^1$ sends $\perp$ to its CF-SFE TTP, $\mathcal{S}^j$ sends $\perp$ to its own trusted party leading to an abort of the execution. Otherwise, $\mathcal{S}^j$ receives $x_i'^j, r_i'^j$ from $\mathcal{S}_{\mathsf{cf}}^1$ and computes $\gamma_i'^j = \mathsf{C}_i(x_i'^j; r_i'^j)$. It also samples a random $x_{3-i}'^j \in \mathcal{M}_{3-i}$, computes $\gamma_{3-i}'^j = \mathsf{C}_{3-i}(x_{3-i}'^j; r_{3-i}'^j)$ using uniform randomness $r_{3-i}'^j \in \mathcal{R}_{3-i}$, and sends the result to $\mathcal{A}^j$.

3. $\mathcal{S}^j$ sends $x_i'^j$ to its TTP, and receives $(y_i^j = f_i(x_1'^j, x_2'^j), \sigma_{3-i})$ back, where $x_{3-i}'^j = x_{3-i}^j$. Recall that σ_{3-i} shows the number of distinct inputs used by the honest party P_{3-i}. If σ_{3-i} has been incremented since the last execution (this information is passed from $\mathcal{S}^{j-1}$ to $\mathcal{S}^j$), then $\mathcal{S}^j$ updates the state to $\Gamma_i := \Gamma_i \cup \{\gamma_{3-i}'^j\}$. Otherwise, it internally runs the ZK simulator $\mathcal{S}_{\mathsf{zk}}(\gamma_{3-i}'^j)$ proving to $\mathcal{A}^j$ that $\gamma_{3-i}'^j \in L_{3-i}$. (Note that the last step involves the state Γ_i.) Notice that $\mathcal{S}_{\mathsf{zk}}$ may itself need to rewind $\mathcal{A}^j$. However, this is not an issue because our simulator $\mathcal{S}$ invokes different simulators sequentially; in particular, $\mathcal{A}^j$ will be in a consistent state when $\mathcal{S}_{\mathsf{zk}}$ terminates and thus $\mathcal{S}$ can proceed with the rest of the simulation.

   $\mathcal{S}^j$ also plays the role of the verifier in the zero-knowledge protocol (with $\mathcal{A}^j$ being the prover). If the value ε is received, or in case the corresponding input $x_i'^j$ is not used in one of the previous executions (note that $\mathcal{S}^j$ can determine this by inspecting Γ_i), then $\mathcal{S}^j$ updates the state to $\Gamma_i := \Gamma_i \cup \{(x_i'^j, r_i'^j)\}$. Otherwise, the state is not modified. (Note that at this stage $\mathcal{S}^j$ is not updating the state on the basis of the verification of the proof itself.)

4. Finally, $\mathcal{S}^j$ invokes $\mathcal{S}_{\mathsf{cf}}^2$ on input $(x_i'^j, r_i'^j, \gamma_{3-i}'^j)$; the simulator $\mathcal{S}_{\mathsf{cf}}^2$ itself runs $\mathcal{A}^j$ who controls party P_i in π_f^2. We emphasize $\mathcal{S}_{\mathsf{cf}}^2$ does not run a new instance of $\mathcal{A}^j$ but it continues running the same instance that has been running so far. If $\mathcal{S}_{\mathsf{cf}}^2$ sends $\perp$, $\mathcal{S}^j$ sends $\perp$ to its trusted party leading to an abort of the execution. Else, $\mathcal{S}_{\mathsf{cf}}^2$ sends the $\mathsf{continue}$ flag. $\mathcal{S}^j$ replies (on behalf of the CF-SFE TTP) by sending to $\mathcal{S}_{\mathsf{cf}}^2$, the output y_i^j it obtained earlier in the simulation.

5. Afterwards, $\mathcal{S}^j$ passes y_i^j to $\mathcal{A}^j$ and outputs whatever $\mathcal{A}^j$ does.

We now need to show that $\text{IDEAL}_{f,\mathcal{S}(\mathbf{z}),i}^{\hbar\text{-rr}}(\mathbf{x}_i, \mathbf{x}_{3-i}, \lambda, \ell) \equiv_c \text{REAL}_{\widehat{\pi}_f,\mathcal{A}(\mathbf{z}),i}^{\hbar}(\mathbf{x}_i, \mathbf{x}_{3-i}, \lambda, \ell)$. By a standard hybrid argument (losing a factor $1/\ell$ in the computational distance), it suffices to show indistinguishability for a single execution (i.e., the jth execution). Therefore, we need to show that for all $i \in \{1, 2\}$ and $j \in [\ell]$

$$\text{IDEAL}_{f,\mathcal{S}(z^j),i}^{\hbar\text{-rr}}(x_i^j, x_{3-i}'^j, \lambda)_j \equiv_c \text{REAL}_{\widehat{\pi}_f,\mathcal{A}(z^j),i}^{\hbar}(x_i^j, x_{3-i}^j, \lambda)_j.$$

We consider a series of intermediate hybrid experiments.

Hybrid $\mathbf{H}\text{YB}_{\mathcal{A}(z^j)}^1(x_i'^j, x_{3-i}'^j, \lambda)_j$: In the first hybrid experiment, we replace $\mathcal{S}^j$ by $\mathcal{S}_1^j$ who controls P_i in the ideal world. Essentially $\mathcal{S}_1^j$ is different from $\mathcal{S}^j$ merely in the way it updates the state Γ_i. Namely, instead of looking at the output σ_{3-i} and checking that $x_i'^j$ is not used in one of the previous executions (cf. item 3. in the description of $\mathcal{S}^j$), it first plays the role of the verifier in the argument system as party P_{3-i} would do in a real execution of the protocol. Hence, if the verification fails or the value ε is received, the state is updated as $\Gamma_i := \Gamma_i \cup \{(x_i'^j, r_i'^j)\}$. The verification process is also applied to the proof returned by the ZK simulator $\mathcal{S}_{\mathsf{zk}}$, and the value $\gamma_{3-i}'^j$ is eventually added to the state depending on the outcome of the verification procedure. Otherwise, the state is not modified. Everything else is identical to the previous simulation. Next, we argue that

$$\text{IDEAL}^{\hbar\text{-}\mathsf{rr}}_{f,\mathcal{S}(z^j),i}(x_i'^j, x_{3-i}'^j, \lambda)_j \equiv_c \text{HYB}^1_{f,\mathcal{A}(z^j),i}(x_i'^j, x_{3-i}'^j, \lambda)_j.$$

In fact, the modification above only affects the way Γ_i is updated. Denote with bad the event that $\mathcal{A}$ is able to come up with an accepting proof for a false statement, i.e., $\mathcal{A}$ is able to convince P_{3-i} that a *fresh* input is equal to one of the previously used inputs. Note that the distribution produced by the two experiments above is identical provided that bad does *not* happen. Due to the soundness property of the ZK argument system, we must conclude that bad happens at most with negligible probability, thus showing that the two experiments are computationally indistinguishable.

Hybrid $\textbf{HYB}^2_{\mathcal{A}(z^j)}(x_i'^j, x_{3-i}^j, \lambda)_j$: In the second hybrid experiment, we replace $\mathcal{S}_1^j$ by $\mathcal{S}_2^j$ who controls P_i in the ideal world and at the same time plays the role of the TTP playing all the roles by itself. As a result, $\mathcal{S}_2^j$ directly interacts with P_{3-i} during the ideal execution of the commit-first protocol. Essentially, $\mathcal{S}_2^j$ is identical to $\mathcal{S}_1^j$ with the exception that it is able to compute and send the correct commitment $\gamma_{3-i}^j = \mathsf{C}_{3-i}(x_{3-i}^j; r_{3-i}^j)$ to $\mathcal{A}^j$. Also, $\mathcal{S}_2^j$ needs to simulate the values (σ_i, σ_{3-i}) by itself. This is done as it would be done in a real execution of the protocol. More precisely, σ_i (resp. σ_{3-i}) is modified based on the verification of the ZK argument for language L_i (resp. L_{3-i}). Here, the simulator will also check whether the values σ_1, σ_2 exceed the rates $\hbar_1, \hbar_2$ and output $\perp$ if so. Finally, note that $\mathcal{S}^2_{\mathsf{cf}}$ is now invoked on the correct inputs, i.e., x_{3-i}^j and γ_{3-i}^j. Everything else is analogous to the previous simulation. Next, we argue that

$$\text{HYB}^1_{f,\mathcal{A}(z^j),i}(x_i'^j, x_{3-i}'^j, \lambda)_j \equiv_c \text{HYB}^2_{f,\mathcal{A}(z^j),i}(x_i'^j, x_{3-i}^j, \lambda)_j.$$

We first argue that the simulation of the values (σ_i, σ_{3-i}) is perfect. This is immediate for the rate of the honest party σ_{3-i} (as the honest party always produces a valid proof, which is thus accepting by completeness of the argument system). As for the rate σ_i corresponding to the corrupted party, it is sufficient to observe that the perfectly binding property of C_i implies that the function f is evaluated on the very same input extracted by the simulator. It follows that the only difference between the previous hybrid and the hybrid world described above is that the real input of the honest party is used in the latter. In particular, $\mathcal{S}_2^j$ feeds both $\mathcal{S}^1_{\mathsf{cf}}$ and $\mathcal{S}^2_{\mathsf{cf}}$ with the commitment $\mathsf{C}_{3-i}(x_{3-i}^j; r_{3-i}^j)$ to the real input x_{3-i}^j as opposed to an arbitrary input $x_{3-i}'^j$.

Given a PPT distinguisher $\mathcal{D}$ between the two hybrids, we can easily construct a PPT distinguisher $\mathcal{D}_{\mathsf{com}}$ breaking the hiding property of the commitment scheme. $\mathcal{D}_{\mathsf{com}}$ takes as input a commitment γ^*—which is either a commitment to x_{3-i}^j or to a random $x_{3-i}'^j$—and perfectly emulates $\mathcal{S}_2^j$ except that it uses γ^* in place of γ_{3-i}^j. At the end of the simulation $\mathcal{D}_{\mathsf{com}}$ outputs the same as $\mathcal{D}$. This results in a perfect simulation, thus establishing the computational indistinguishability of the two hybrids.

Note that we rely here on the fact that, in both worlds, the simulator emulates the ZK proof using $\mathcal{S}_{\mathsf{zk}}$ as opposed to executing the real proof. In particular, the ZK simulator does not need the parties' private inputs for its simulation, and hence is not affected by the aforementioned change in the inputs.

Hybrid $\textsc{Hyb}^3_{\mathcal{A}(z^j)}(x_i'^j, x_{3-i}^j, \lambda)$: We modify the previous hybrid world, by having $\mathcal{S}_3^j$ provide an actual ZK proof that an input is re-used from a previous execution or to send an empty string ε if this is not case. For this purpose, $\mathcal{S}_3^j$ uses the state (Γ_1, Γ_2) and the current inputs x_1^j, x_2^j. The zero-knowledge property of the proof system automatically guarantees that the view generated using the real proof and the simulator $\mathcal{S}_{\mathsf{zk}}$ are computationally indistinguishable which in turn implies the computational indistinguishability of the output of the current hybrid experiment and the previous one. Thus, we have

$$\textsc{Hyb}^2_{f,\mathcal{A}(z^j),i}(x_i'^j, x_{3-i}^j, \lambda)_j \equiv_c \textsc{Hyb}^3_{f,\mathcal{A}(z^j),i}(x_i'^j, x_{3-i}^j, \lambda)_j.$$

To conclude the proof, it suffices to note that $\textsc{Hyb}^3_{f,\mathcal{A}(z^j),i}(x_i'^j, x_{3-i}^j, \lambda)_j$ exactly equals the output distribution of $\mathcal{A}^j$ in the real world, thus proving that $\widehat{\pi}_f \leftarrow \Psi_{\mathsf{rr}}(\pi_f, \mathscr{r}_1, \mathscr{r}_2, \ell)$ is a secure $\mathsf{rate\text{-}revealing}$ $(\mathscr{r}_1, \mathscr{r}_2)$-limited protocol for function f.

□

5.3 A Pattern-Revealing Compiler

In this section, we introduce a more efficient compiler Ψ_{pr} for designing rate-limited SFE, a detailed description of which appears in Fig. 3. Given as input a CF-SFE protocol, our compiler Ψ_{pr} outputs a weaker form of rate-limited SFE where each party not only learns the current rate for its counterpart during each execution, but also the pattern of already used inputs (see Section 4). The main advantage is that this new compiler adds very little overhead to the original CF-SFE.

The overview. The idea is as follows. Besides their input, each party also stores a secret key for a PRF (with a different key for each party). Before invoking the commit-first SFE protocol, each player generates the randomness it needs for the committing phase by applying the PRF on the *chosen input* for this execution. With this modification in place, the committing phase for each party becomes deterministic. If a party uses the same input in two executions, the two commitments its counterpart receives will be identical. As a result, to check a repeated input, each party can compare the commitment for the current execution with those used in the previous ones, and determine if the input is new or being repeated (hence also learning the pattern of used inputs). Note that the commitments still provide the required hiding and binding properties. The only overhead imposed by this compiler is the application of a PRF to generate the randomness for the committing phase.

A careful reader might observe that a malicious party is not obliged to use the PRF, and thus it could either repeat a previously used input while changing the commitment or change the input while repeating a previously used commitment. Note, however, that the former case does not constitute an attack, as the adversary is actually decreasing its own rate even when using a repeated input.[5] The latter case, instead, contradicts the binding property of the commitment scheme.

Theorem 5.3. *Let $\pi_f = (\pi_f^1, \pi_f^2)$ be a commit-first SFE (w.r.t. $\mathsf{C}_1, \mathsf{C}_2$) securely evaluating function $f = (f_1, f_2)$, and assume that $\mathsf{C}_1, \mathsf{C}_2$ are perfectly binding and computationally hiding commitment schemes, and that* PRF *is a pseudo-random function. Then $\widehat{\pi}_f \leftarrow \Psi_{\mathsf{pr}}(\pi_f, \mathscr{r}_1, \mathscr{r}_2, \ell)$ of Fig. 3 is a secure* $\mathsf{pattern\text{-}revealing}$ *$(\mathscr{r}_1, \mathscr{r}_2)$-limited SFE for f.*

[5] Looking ahead, this behavior can be simulated by extracting the malicious party's input and randomness.

Pattern-Revealing Compiler Ψ_{pr}:

Given as input a commit-first protocol (w.r.t. $\mathsf{C}_1, \mathsf{C}_2$) $\pi_f = (\pi_f^1, \pi_f^2)$, a rate $\mathscr{r} = (\mathscr{r}_1, \mathscr{r}_2)$, and a number of executions ℓ, the compiled protocol $\widehat{\pi}_f$ consists of three phases, described below. Party P_1 and P_2 hold private keys k_1 and k_2 (for a PRF) and the state variables $\Gamma_1, \Gamma_2 := \emptyset$, respectively. For each execution $j \in [\ell]$, $\widehat{\pi}_f$ proceeds as follows:

Randomness Generation Phase: Parties P_1 and P_2, holding respectively inputs x_1^j and x_2^j compute values $r_1^j := \mathsf{PRF}(k_1, x_1^j)$ and $r_2^j := \mathsf{PRF}(k_2, x_2^j)$, respectively.

Committing Phase: Parties P_1 and P_2, run the protocol π_f^1 on the same inputs as in the first phase yielding the output $((x_1^j, r_1^j, \gamma_2^j = \mathsf{C}_2(x_2^j; r_2^j)), (x_2^j, r_2^j, \gamma_1^j = \mathsf{C}_1(x_1^j; r_1^j)))$ where r_1^j, and r_2^j are from the first phase. If $\gamma_{3-i}^j \notin \Gamma_i$, party P_i adjusts his rate by letting $\mathscr{r}_i = \mathscr{r}_i - 1$. Party P_i then updates its state $\Gamma_i = \Gamma_i \cup \{(\gamma_{3-i}^j, j)\}$. If $\mathscr{r}_i$ equals 0, abort the execution.

Protocol Emulation Phase: P_1 and P_2 run the protocol π_f^2 on the same inputs as in the first phase, yielding the output (y_1^j, y_2^j).

Figure 3: A compiler for pattern-revealing rate-limited SFE.

Proof. Similar to the proof of Theorem 5.2, we consider an adversary $\mathcal{A} = (\mathcal{A}^1, \ldots, \mathcal{A}^\ell)$ corrupting party P_i during the ℓ executions of $\widehat{\pi}_f$ where $\mathcal{A}^j$ represents $\mathcal{A}$'s strategy during the j-th execution. Again, we denote by $\mathrm{REAL}^{\mathscr{r}\text{-}\mathsf{pr}}_{\widehat{\pi}_f, \mathcal{A}(\mathbf{z}), i}(\mathbf{x}_1, \mathbf{x}_2, \lambda, \ell)$ the joint distribution of the output of all the $\mathcal{A}^j$s combined.

We describe a simulator $\mathcal{S} = (\mathcal{S}^1, \ldots, \mathcal{S}^\ell)$ in the ideal world that mimics $\mathcal{A}$'s output. Our simulator makes use of the simulator $\mathcal{S}_{\mathsf{cf}} = (\mathcal{S}^1_{\mathsf{cf}}, \mathcal{S}^2_{\mathsf{cf}})$ which exists due to the fact that our compiler takes as input a commit-first SFE protocol π_f. The simulator $\mathcal{S} = (\mathcal{S}^1, \ldots, \mathcal{S}^\ell)$ runs a copy of $\mathcal{A}$, and keeps a state Γ_i and a list Γ'. Both Γ_i and Γ' are initially set to be empty. The j-th execution of $\mathcal{S}$ is given below.

1. $\mathcal{S}^j$ takes $(x_i^j, \Gamma_i, \Gamma', z^j)$ as input.

2. In the randomness-generation phase, $\mathcal{S}^j$ does nothing.

3. In the committing phase, $\mathcal{S}^j$ invokes $\mathcal{S}^1_{\mathsf{cf}}$ on input x_i^j. The simulator $\mathcal{S}^1_{\mathsf{cf}}$ invokes $\mathcal{A}^j$ who controls party P_i in π_f^1. If $\mathcal{S}^1_{\mathsf{cf}}$ sends $\perp$ to its CF-SFE TTP, $\mathcal{S}^j$ sends $\perp$ to its own trusted party leading to an abort of the execution. Otherwise, $\mathcal{S}^j$ receives $x_i'^j, r_i'^j$ from $\mathcal{S}^1_{\mathsf{cf}}$.

4. $\mathcal{S}^j$ sends $x_i'^j$ to its TTP, and receives $(y_i^j = f_i(x_1'^j, x_2'^j), J_{3-i})$ back, where $x_{3-i}'^j = x_{3-i}^j$. Recall that J_{3-i} indicates an index $J_{3-i} < j$ of the first execution with its counterpart P_{3-i} where P_{3-i} used the same input as in this jth execution. Let us denote $t := J_{3-i}$.

5. Now, $\mathcal{S}^j$ computes $\gamma_i'^j = \mathsf{C}_i(x_i'^j; r_i'^j)$. If $(\gamma_i'^j, *) \notin \Gamma_i$ then $\mathcal{S}^j$ updates the state Γ_i by letting $\Gamma_i := \Gamma_i \cup (\gamma_i'^j, j)$.

   If $t < j$, $\mathcal{S}^j$ sets $x_{3-i}'^j := x_{3-i}'^t$ for input $x_{3-i}'^t$ previously chosen in the t-th execution, looks for the element $(x_{3-i}'^j, r') \in \Gamma'$, and computes $\gamma_{3-i}'^j = \mathsf{C}_{3-i}(x_{3-i}'^j; r')$; else it computes $\gamma_{3-i}'^j = \mathsf{C}_{3-i}(x_{3-i}'^j; r_{3-i}'^j)$ using uniformly sampled input $x_{3-i}'^j \in \mathcal{M}_{3-i}$ and uniform randomness $r_{3-i}'^j \in \mathcal{R}_{3-i}$. Thus, Γ' (resp. Γ_{3-i}) is updated to $\Gamma' := \Gamma' \cup (x_{3-i}'^j; r_{3-i}'^j)$ (resp. $\Gamma_{3-i} := \Gamma_{3-i} \cup (\gamma_{3-i}'^j, j)$), and $\mathcal{S}^j$ sends $\gamma_{3-i}'^j$ to $\mathcal{A}^j$.

6. Finally, $\mathcal{S}^j$ invokes $\mathcal{S}^2_{\mathsf{cf}}$ on input $(x_i'^j, \gamma_{3-i}'^j)$; $\mathcal{S}^2_{\mathsf{cf}}$ itself runs $\mathcal{A}^j$ who controls party P_i in π_f^2. If $\mathcal{S}^2_{\mathsf{cf}}$ sends $\perp$, $\mathcal{S}^j$ sends $\perp$ to its trusted party leading to an abort of the execution. Else,

$\mathcal{S}_{\mathsf{cf}}^2$ sends the $\mathsf{continue}$ flag. $\mathcal{S}^j$ replies (on behalf of the CF-SFE TTP) by sending to $\mathcal{S}_{\mathsf{cf}}^2$, the output y_i^j he obtained earlier in the simulation.

7. Afterwards, $\mathcal{S}^j$ passes y_i^j to $\mathcal{A}^j$ and outputs whatever $\mathcal{A}^j$ does.

We now need to show that $\text{IDEAL}^{\mathcal{R}\text{-}\mathsf{pr}}_{f,\mathcal{S}(\mathbf{z}),i}(\mathbf{x}_i, \mathbf{x}_{3-i}, \lambda, \ell) \equiv_c \text{REAL}^{\mathcal{R}}_{\widehat{\pi}_f,\mathcal{A}(\mathbf{z}),i}(\mathbf{x}_i, \mathbf{x}_{3-i}, \lambda, \ell)$. By a standard hybrid argument (losing a factor $1/\ell$ in the computational distance), it suffices to show indistinguishability for a single execution (i.e., the jth execution). Therefore, we need to show that for all $i \in \{1, 2\}$ and $j \in [\ell]$

$$\text{IDEAL}^{\mathcal{R}\text{-}\mathsf{pr}}_{f,\mathcal{S}(z^j),i}(x_i'^j, x_{3-i}'^j, \lambda)_j \equiv_c \text{REAL}^{\mathcal{R}}_{\widehat{\pi}_f,\mathcal{A}(z^j),i}(x_i'^j, x_{3-i}^j, \lambda)_j.$$

We consider two intermediate hybrid experiments. In the first experiment, we modify the simulator by generating the random coins for the commitments by an application of a PRF on the input. We argue that this modification is not distinguishable by the adversary $\mathcal{A}^j$ due to the pseudo-randomness property of the PRF. In the second experiment, we assume that in contrast to the simulation above, the real input of the honest party is used in the simulation. We argue that this modification is not distinguishable by the adversary $\mathcal{A}^j$ due to the hiding property of the commitment scheme. It is easy to see that the distribution of $\mathcal{A}^j$'s output in the last experiment is identical to the distribution of its output in the real protocol, which concludes our proof. Details follow.

Hybrid $\textbf{H}\text{YB}^1_{\mathcal{A}(z^j)}(x_i'^j, x_{3-i}'^j, \lambda)_j$: In the first hybrid experiment, we replace $\mathcal{S}^j$ by $\mathcal{S}_1^j$ who controls P_i in the ideal world. Essentially, $\mathcal{S}_1^j$ is different from $\mathcal{S}^j$ merely in one point. Instead of computing the commitment $\gamma_{3-i}'^j = \mathsf{C}_{3-i}(x_{3-i}'^j; r_{3-i}^j)$—which in the simulation belongs to the honest party—using a uniformly sampled randomness r_{3-i}^j, it computes r_{3-i}^j by using a PRF (cf. item 5 in the description of $\mathcal{S}^j$). More precisely, $\mathcal{S}_1^j$ computes $r_{3-i}^j := \mathsf{PRF}(k_{3-i}, x_{3-i}'^j)$ using secret key $k_{3-i} \in \mathcal{K}$ which is sampled initially by the simulator. Note that we explicitly do not instantiate the randomness of the malicious party P_i by a PRF since in the real world the party could use possibly non-uniform randomness.

Next, we argue that

$$\text{IDEAL}^{\mathcal{R}\text{-}\mathsf{pr}}_{f,\mathcal{S}(z^j),i}(x_i'^j, x_{3-i}'^j, \lambda)_j \equiv_c \text{HYB}^1_{f,\mathcal{A}(z^j),i}(x_i'^j, x_{3-i}'^j, \lambda)_j,$$

which intuitively follows by security of the PRF. In fact, the modification above only affects the way the random coins of the commitments for the honest party are generated. More precisely, given a PPT distinguisher $\mathcal{D}$ telling apart the two experiments we construct another PPT distinguisher $\mathcal{D}_{\mathsf{prf}}$ breaking security of the PRF as follows. $\mathcal{D}_{\mathsf{prf}}$ behaves exactly as $\mathcal{S}_1^j$, except when it has to compute the commitment $\gamma_{3-i}'^j$ which is now generated by first forwarding the value $x_{3-i}'^j$ to the target oracle and then using as randomness the value r_{3-i}^j received from the oracle. Finally, at the end of the last execution, $\mathcal{D}_{\mathsf{prf}}$ outputs the same as $\mathcal{D}$.

Notice that the above simulation is perfect, in that depending on $\mathcal{D}_{\mathsf{prf}}$'s oracle being a truly random function or a PRF, the simulation yields exactly the same distribution as in the ideal experiment or in the first hybrid experiment (respectively). Thus, the two experiments are computationally close.

Hybrid $\textbf{H}\text{YB}^2_{\mathcal{A}(z^j)}(x_i'^j, x_{3-i}^j, \lambda)_j$: In the second hybrid experiment, we replace $\mathcal{S}_1^j$ by $\mathcal{S}_2^j$ who controls P_i in the ideal world and at the same time plays the role of the TTP. As a result,

$\mathcal{S}_2^j$ directly interacts with P_{3-i} during the ideal execution of the commit-first protocol. Essentially, $\mathcal{S}_2^j$ is identical to $\mathcal{S}_1^j$ with the exception that it is able to compute and send the correct commitment $\gamma_{3-i}^j = \mathsf{C}_{3-i}(x_{3-i}^j; r_{3-i}^j)$.

$\mathcal{S}_2^j$ needs also to simulate the values (J_i, J_{3-i}) by itself. This is done as it would be done in a real execution of the protocol. More precisely, J_i (resp. J_{3-i}) is the second entry of $(\gamma_i, *) \in \Gamma_i$ (resp. $(\gamma_{3-i}, *) \in \Gamma_{3-i}$). Note that an entry $(\gamma_i, *)$ must be in Γ_i since $\mathcal{S}_1^j$, and thus $\mathcal{S}_2^j$, updates the state Γ_i consistently.

The simulator will also check whether the values J_1, J_2 exceed the rates κ_1, κ_2 and output $\perp$ if so. Finally, note that $\mathcal{S}_{\mathsf{cf}}^2$ is now invoked on the correct inputs, i.e., x_{3-i}^j and γ_{3-i}^j. Everything else is analogous to the previous simulation.

Next, we argue that

$$\textsc{Hyb}^1_{f,\mathcal{A}(z^j),i}(x_i'^j, x_{3-i}'^j, \lambda)_j \equiv_c \textsc{Hyb}^2_{f,\mathcal{A}(z^j),i}(x_i'^j, x_{3-i}^j, \lambda)_j.$$

We first argue that the simulation of the values (J_i, J_{3-i}) is perfect. This is immediate for the index of the honest party J_{3-i} (as the honest party always computes the randomness for the commitment by invoking the PRF on its own input). As for the index J_i corresponding to the corrupted party, it is sufficient to observe that the perfectly binding property of C_i implies that the function f is evaluated on the very same input extracted by the simulator. It follows that the only difference between the previous hybrid and the hybrid world described above is that the real input of the honest party is used in the latter. In particular, $\mathcal{S}_2^j$ feeds both $\mathcal{S}_{\mathsf{cf}}^1$ and $\mathcal{S}_{\mathsf{cf}}^2$ with the commitment $\mathsf{C}_{3-i}(x_{3-i}^j; r_{3-i}^j)$ to the real input x_{3-i}^j as opposed to a random input $x_{3-i}'^j$.

Given a PPT distinguisher $\mathcal{D}$ between the two hybrids, we can easily construct a PPT distinguisher $\mathcal{D}_{\mathsf{com}}$ breaking the hiding property of the commitment scheme. $\mathcal{D}_{\mathsf{com}}$ takes as input a commitment γ^*—which is either a commitment to x_{3-i}^j or to a random $x_{3-i}'^j$—and perfectly emulates $\mathcal{S}_2^j$ except that it uses γ^* in place of γ_{3-i}^j. At the end of the simulation $\mathcal{D}_{\mathsf{com}}$ outputs the same as $\mathcal{D}$. This results in a perfect simulation, thus establishing the computational indistinguishability of the two hybrids.

To conclude the proof, it suffices to note that $\textsc{Hyb}^2_{f,\mathcal{A}(z^j),i}(x_i'^j, x_{3-i}^j, \lambda)_j$ exactly equals the output distribution of $\mathcal{A}^j$ in the real world, thus proving that $\widehat{\pi}_f \leftarrow \Psi_{\mathsf{pr}}(\pi_f, \kappa_1, \kappa_2, \ell)$ is a secure pattern-revealing (κ_1, κ_2)-limited protocol for function f. □

6 Making the Compilers Stateless

One drawback of the compilers described in the previous section is that both P_1 and P_2 need to maintain state. To some extent, this assumption is necessary. It is not too hard to see that RL-SFE is impossible to achieve if neither party keeps any state information about previous executions. However, as discussed earlier, in many natural client-server applications of SFE in the real world, it is reasonable to assume that the servers keep state, while the clients typically do not.

In this section, we show how to modify the compilers from Section 5 in such a way that only one of the parties needs to keep state. Our solution is efficient and works for all three compilers we discussed earlier. Throughout this section, we assume P_1 is the client and P_2 is the server. Server P_2 receives no output (as it is usually the case in the client-server setting) and wants to enforce the rate limit κ for the client. Although P_1 does not maintain any state, it needs to make sure that P_2 handles the rate honestly. On the other hand, the server also needs to be convinced that the client is not cheating, by exceeding the rate limit $\mathscr{r}$.

The overview. Note that in the stateful versions of our compilers, P_1 needs to keep state in order to generate a ZK proof of repeated inputs, and verify the corresponding statement being proven by P_2. Since we are only enforcing the rate for P_1, we can eliminate the latter ZK proofs, and focus on the first one. Although our approach is general, for the sake of simplicity, we describe it in relation to our rate-revealing compiler from Section 5.2. The same idea can be applied to make our rate-hiding compiler (cf. Section 5.1) stateless.[6] The basic idea is simple: We ask the server to store the list of all the commitments previously sent by P_1 who sends the list to the client, during each run. For this simple approach to work, we need to address several important issues:

- For the client to learn the current rate and the previously queried inputs before each execution, it needs to store these values on the server side in a secure way. This can be easily addressed by having P_1 encrypt the message and randomness for each commitment (using a symmetric-key encryption scheme) and send it along with the commitment itself. P_1 will just keep the private key for the encryption scheme.

- The client needs to verify that the list of commitments it receives from the server are the original commitments it sent in the previous executions. To do so, in each run P_1 computes a MAC ϕ of the string obtained by hashing all the commitments (i.e., the concatenation of the list it obtains from the server and the one it creates in the current execution) and sends it to the server. In each execution, it requests this MAC, the list of commitments along with the ciphertext storing the inputs and random coins from the server. Due to the unforgeability of the MAC, the server will only be able to use a correct list of commitments, previously issued and MACed by the client itself.

- Clearly, the above described protocol allows the server to cheat and only send a subset of the commitment list, along with a tag generated for that subset in one of the earlier executions, to the client. Since the client does not keep any state, it will not be able to detect this attack. For technical reasons, this will require us to slightly modify the description of the ideal world in the definition of rate-revealing SFE.[7]

  However, a more careful inspection shows that the above issue does not really constitute an attack. In fact, the tag ϕ already *binds* the current rate to the current list of commitments, and, in particular, makes it hard for the server to cook-up a state such that the verification of the tag is successful, and the client will think its rate is already exceeded when it is not. Essentially, coming up with such a state requires to either find a collision in the hash function, or to forge a tag for a fake list of commitments.

A detailed description of the compiler is depicted in Fig. 4.

Let us specify the ideal world for (stateless) rate-revealing SFE a bit more precisely. In case of a corrupted P_1, the ideal distribution is exactly the same as the one defined in Section 4. In case P_2 is corrupted, at each execution $j \in [\ell]$, the ideal adversary $\mathcal{S}^j$ is allowed to specify (along with the modified input $x_2'^j$) an index $t \leq j$; this will cause the TTP to use $\mathcal{X}_1' = \mathcal{X}_1[1, \ldots, t]$ (i.e., the first t elements of $\mathcal{X}_1$) to compute the rate of party P_1 at the j-th execution. We speak of *stateless* rate-revealing SFE.

[6] In the client-server scenario, our pattern-revealing compiler from Section 5.3 is already stateless.

[7] Let us emphasize that we can achieve the "full-fledged" notion of rate-revealing SFE if we let the client store (and update) the hash of the state that was recovered in the last execution. This requires the client to keep (and update) a state of constant size.

Rate-Revealing Compiler Ψ_{rr} (stateless version):

Let $(\mathsf{S}, \mathsf{T}, \mathsf{V})$ be a MAC, $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ be an SKE scheme and $\{H_\iota\}_{\iota \in I}$ be a set of collision-resistant hash functions. Party P_1 stores values (k, s, ι) where $s \leftarrow \mathsf{S}(1^\lambda)$, $k \leftarrow \mathsf{G}(1^\lambda)$, and $\iota \leftarrow \mathsf{I}(1^\lambda)$. Given as input a commit-first protocol (w.r.t. C) $\pi_f = (\pi_f^1, \pi_f^2)$ for function $f = (f_1, -)$, a rate ρ, and a number of executions ℓ, the compiled protocol $\widehat{\pi}_f$ consists of four phases, described below. Party P_2 initializes the state variable $\Sigma := \emptyset$. For each execution $j \in [\ell]$, $\widehat{\pi}_f$ proceeds as follows.

Recovery of State Phase: Party P_1 receives the state $\Sigma = \{\Gamma, \Omega, \phi\}$ from P_2, where

$$\Gamma = \{\gamma^1, \dots, \gamma^{j-1}\} \qquad \Omega = \{(c^1, \bar{c}^1), \dots, (c^{j-1}, \bar{c}^{j-1})\},$$

and ϕ is a tag. Hence, P_1 computes $h = H_\iota(\gamma_1^1, \dots, \gamma_1^{j-1})$ and runs $\mathsf{V}(s, h, \phi)$; if the verification fails, P_1 sends $\perp$ to P_2 and halts the execution. Otherwise, it uses the key k to extract $x_1^i = \mathsf{D}(k, c^i)$ and $r_1^i = \mathsf{D}(k, \bar{c}^i)$ for all $i \in [j-1]$. Letting $\Lambda^{j-1} = \{(x_1^i, r_1^i)\}_{i=1}^u$, where $u \in \mathbb{N}$ denotes the number of *distinct* x_1^i's values, P_1 proceeds to the next step.

Committing Phase: Party P_1 (holding input x_1^j) runs the protocol π_f^1 yielding the output $((x_1^j, r_1^j), \gamma^j = \mathsf{C}(x_1^j; r_1^j))$. It also computes $c^j = \mathsf{E}(k, x_1^j)$, $\bar{c}^j = \mathsf{E}(k, r_1^j)$ and sends the result to P_2.

Proof of Repeated-Input Phase: If x_1^j is indeed being repeated, party P_1 proceeds to give a ZK proof of this fact, using the language L as described in the protocol of Section 5.2. (Notice that this involves the recovered state Λ^{j-1}.) Otherwise, P_1 checks that $u \leq \rho$ and forwards the empty string ε if this is the case. If the rate is exceeded, P_1 outputs $\perp$ and aborts. Provided that it did not abort, P_1 updates the hash value $h := H_\iota(\gamma^1, \dots, \gamma^j)$, computes $\phi = \mathsf{T}(s, h)$ and forwards ϕ to P_2.

Party P_2 verifies the proof and updates the rate ρ as specified in the protocol from Section 5.2. Moreover, it updates Σ by letting $\Gamma := \Gamma \cup \{\gamma^j\}$, $\Omega := \Omega \cup \{(c^j, \bar{c}^j)\}$ and storing the new ϕ.

Protocol Emulation Phase: P_1 and P_2 run the protocol π_f^2 on input x_1^j (the same as in the committing phase) and x_2^j, yielding the output $(y_1^j, -)$.

Figure 4: Stateless version of the rate-revealing compiler Ψ_{rr} from Section 5.

Theorem 6.1. *Let $\pi_f = (\pi_f^1, \pi_f^2)$ be a commit-first protocol (w.r.t. C) securely evaluating function $f = (f_1, -)$, and assume that C is a perfectly binding and computationally hiding commitment scheme, that $(\mathsf{S}, \mathsf{T}, \mathsf{V})$ is* EUF-CMA*, that $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ is* CPA*-secure, that $\{H\}_{\iota \in I}$ is a collision-resistant hash functions family, and that $(\mathcal{P}, \mathcal{V})$ is a ZK argument system for the language L. Then, $\widehat{\pi}_f \leftarrow \Psi_{\mathsf{rr}}(\pi_f, \rho, \ell)$ is a secure* stateless rate-revealing *ρ-limited protocol for f.*

Proof. Since the protocol is asymmetric, we need to deal with the corruption of P_1 and P_2 separately.

The client is corrupted. Assuming P_2 is honest, in each execution $j \in [\ell]$ party P_1 can perfectly reconstruct the state Λ^{j-1} thanks to the secret key k for the SKE scheme. Apart from the way the state is reconstructed, the compiler is identical to the transformation of Section 5.2. Hence, the same simulator $\mathcal{S}$ in the proof of Theorem 5.2 will do here.

The server is corrupted. Consider an adversary $\mathcal{A} = (\mathcal{A}^1, \dots, \mathcal{A}^\ell)$ corrupting party P_2 during the ℓ executions of $\widehat{\pi}_f$. We describe a simulator $\mathcal{S} = (\mathcal{S}^1, \dots, \mathcal{S}^\ell)$ in the ideal world—as discussed in Section 4—that mimics $\mathcal{A}$'s output. $\mathcal{S}$ initially picks $s \leftarrow \mathsf{S}(1^\lambda)$, $k \leftarrow \mathsf{G}(1^\lambda)$, $\iota \leftarrow \mathsf{I}(1^\lambda)$, runs a copy of $\mathcal{A}$, and keeps an array Σ initially set to be empty. The j-th execution is given below.

1. $\mathcal{S}^j$ takes $(\Sigma, s, k, \iota, x_2^j, z^j)$ as input.

2. In the recovery of state phase, $\mathcal{S}^j$ receives $(\Gamma', \Omega', \phi')$ from $\mathcal{A}^j$. Hence, it checks whether there exists an entry in Σ such that $\Sigma[i] = (\Gamma', \Omega', \phi')$, for $i \in [j-1]$. If the check fails, $\mathcal{S}^j$ sends $\perp$ to $\mathcal{A}^j$ and halts the simulation, otherwise, it proceeds to the next step.

3. In the committing phase, $\mathcal{S}^j$ invokes $\mathcal{S}^1_{cf}$ on input x_2^j. The simulator $\mathcal{S}^1_{cf}$ invokes $\mathcal{A}^j$ who controls party P_2 in π^1_f. If $\mathcal{S}^1_{cf}$ sends $\perp$ to its CF-SFE TTP, $\mathcal{S}^j$ sends $\perp$ to its own trusted party leading to an abort of the execution. Otherwise, $\mathcal{S}^j$ receives $x_2'^j, r_2'^j$ from $\mathcal{S}^1_{cf}$.

   $\mathcal{S}^j$ also samples a random $x_1'^j \in \mathcal{M}$, computes $\gamma_1'^j = \mathsf{C}(x_1'^j; r_1'^j)$ using uniform randomness $r_1'^j \in \mathcal{R}$, encrypts $c'^j = \mathsf{E}(k, x_1'^j)$, $\bar{c}'^j = \mathsf{E}(k, r_1'^j)$, lets $\Omega := \Omega \cup \{c'^j, \bar{c}'^j\}$ and sends $(\gamma_1'^j, c'^j, \bar{c}'^j)$ to $\mathcal{A}^j$.

4. Let $\Gamma' = \{\gamma_1'^1, \ldots, \gamma_1'^t\}$, for some $t \leq j-1$. $\mathcal{S}^j$ sends $(x_2'^j, t)$ to its TTP. If the TTP returns $\perp$, meaning that the rate is exceeded, the simulator sends also $\perp$ to $\mathcal{A}^j$ and aborts. Otherwise, $\mathcal{S}^j$ receives $(-, \sigma^j)$ back.

   If σ^j has been incremented w.r.t. σ^t (this information is passed from $\mathcal{S}^t$ to $\mathcal{S}^j$), then $\mathcal{S}^j$ sends to $\mathcal{A}^j$ the value ε. Otherwise, the simulator invokes the ZK simulator for the language

   $$L = \{\gamma \in \mathcal{C} : \exists (x_1'^j, r_1'^j, r_1''^j) \text{ s.t. } \gamma_1'^j = \mathsf{C}(x_1'^j; r_1'^j) \wedge \mathsf{C}(x_1'^j; r_1''^j) \in \Gamma'\},$$

   with $\mathcal{A}^j$ acting as the verifier. Finally, $\mathcal{S}^j$ computes the value $\phi' \leftarrow \mathsf{T}(s, H_\iota(\gamma_1'^1, \ldots, \gamma_1'^t, \gamma_1'^j))$. The value ϕ' is forwarded to $\mathcal{A}^j$ and the state is updated as $\Gamma := \Gamma \cup \{\gamma'^j\}$ and $\Sigma[j] := \{\Gamma, \Omega, \phi\}$.

5. $\mathcal{S}^j$ invokes $\mathcal{S}^2_{\mathsf{cf}}$ on input $(x_2'^j, r_2'^j, \gamma_1'^j)$; $\mathcal{S}^2_{\mathsf{cf}}$ itself runs $\mathcal{A}^j$ who controls party P_2 in π^2_f. We emphasize $\mathcal{S}^2_{\mathsf{cf}}$ does not run a new instance of $\mathcal{A}^j$ but it continues with running the same instance that has been running so far. If $\mathcal{S}^2_{cf}$ sends $\perp$, $\mathcal{S}^j$ sends $\perp$ to its trusted party leading to an abort of the execution. Else, $\mathcal{S}^2_{cf}$ sends the $\mathsf{continue}$ flag. $\mathcal{S}^j$ replies (on behalf of the CF-SFE TTP) by sending the empty string to $\mathcal{S}^2_{cf}$.

6. Finally, $\mathcal{S}^j$ outputs whatever $\mathcal{A}^j$ does.

We need to argue that for all $j \in [\ell]$

$$\text{IDEAL}^{\mathcal{R}\text{-}\mathsf{rr}}_{f, \mathcal{S}(z^j), 2}(x_1'^j, x_2'^j, \lambda)_j \equiv_c \text{REAL}^{\mathcal{R}}_{\widehat{\pi}_f, \mathcal{A}(z^j), 2}(x_1^j, x_2'^j, \lambda)_j.$$

We consider a series of intermediate hybrid experiments.

Hybrid $\textsc{Hyb}^1_{\mathcal{A}(z^j)}(x_1'^j, x_2'^j, \lambda)_j$: In the first hybrid experiment, we replace $\mathcal{S}^j$ by $\mathcal{S}^j_1$ who controls P_2 in the ideal world. $\mathcal{S}^j_1$ is identical to $\mathcal{S}^j$, with the only difference that it verifies the state $(\Gamma', \Omega', \phi')$ as P_1 would do in a real execution. Namely, upon input $(\Gamma', \Omega', \phi')$, the simulator computes $h' = H_\iota(\gamma'^1, \ldots, \gamma'^t)$ and runs $\mathsf{V}(s, h', \phi')$. If the verification fails, $\mathcal{S}^j_1$ sends $\perp$ to $\mathcal{A}^j$ and halts the simulation; otherwise, it continues the simulation as $\mathcal{S}^j$ would do. Next, we show that

$$\text{IDEAL}^{\mathcal{R}\text{-}\mathsf{rr}}_{f, \mathcal{S}(z^j), 2}(x_1'^j, x_2'^j, \lambda)_j \equiv_c \textsc{Hyb}^1_{\mathcal{A}(z^j)}(x_1'^j, x_2'^j, \lambda)_j.$$

In fact, there is a difference in the two hybrid experiments only when either of the following bad events happen:

- bad_1: The event becomes true whenever $\mathcal{A}^j$ finds a collision for some of the hash values h';

- bad_2: The event becomes true whenever $\mathcal{A}^j$ forges a tag ϕ' for some Γ' which is different than the Γ' generated by the honest client.

Let $\mathsf{bad} = \mathsf{bad}_1 \vee \mathsf{bad}_2$. It is easy to see that the two hybrids are identically distributed provided that bad does not happen. By a union bound it is thus sufficient to bound the probability of events bad_1 and bad_2. Clearly, any PPT distinguisher provoking event bad_1 with non-negligible probability can be turned into a PPT adversary breaking collision resistance of the hash function with roughly the same probability; thus bad_1 only happens with negligible probability. Similarly, any PPT distinguisher provoking event bad_2 with non-negligible probability can be turned into a PPT adversary breaking EUF-CMA of the MAC with roughly the same probability; thus bad_2 only happens with negligible probability. We conclude that the two experiments are computationally close.

Hybrid $\mathbf{H}\textsc{yb}^2_{\mathcal{A}(z^j)}(x_1^j, x_2'^j, \lambda)_j$: In the second hybrid experiment, we replace $\mathcal{S}_1^j$ by $\mathcal{S}_2^j$ who controls P_2 in the ideal world and at the same time plays the role of the TTP playing all the roles by itself. As a result, $\mathcal{S}_2^j$ directly interacts with P_1 during the ideal execution of the commit-first protocol. Essentially, $\mathcal{S}_2^j$ is identical to $\mathcal{S}_1^j$ with the exception that it is able to compute and send the correct commitment $\gamma_1^j = \mathsf{C}(x_1^j; r_1^j)$ and the right ciphertext $c'^j = \mathsf{E}(k, x_1^j)$ to $\mathcal{A}^j$. Also, $\mathcal{S}_2^j$ needs to simulate the value σ^j by itself. This is done as it would be done in the real protocol, by extracting the inputs in Λ^{j-1} with the help of the secret key k and verifying the proof of $\mathcal{A}^j$. Finally, note that $\mathcal{S}_{cf}^2$ is now invoked on the correct inputs, i.e., x_1^j and γ_1^j. Everything else is analogous to the previous simulation.

Next, we claim that

$$\textsc{Hyb}^1_{\mathcal{A}(z^j)}(x_1'^j, x_2'^j, \lambda)_j \equiv_c \textsc{Hyb}^2_{\mathcal{A}(z^j)}(x_1^j, x_2'^j, \lambda)_j.$$

We first argue that the simulation of the value σ^j is perfect. This is because the perfectly binding property of C implies that the function f is evaluated on the very same input extracted by the simulator. It follows that the only difference between the previous hybrid and the hybrid world described above is that the real input of the honest party is used in the latter. In particular, $\mathcal{S}_2^j$ feeds both $\mathcal{S}^1_{\mathsf{cf}}$ and $\mathcal{S}^2_{\mathsf{cf}}$ with the commitment $\mathsf{C}(x_2^j; r_2^j)$ to the real input x_2^j as opposed to an arbitrary input $x_2'^j$; analogously the ciphertext c'^j contains an encryption of x_2^j.

However, due to the hiding property of the commitment and the CPA-security of the SKE scheme, these two views are computationally indistinguishable. (In particular any distinguisher between the two experiments can be turned into an adversary breaking either the hiding property of the commitment or the CPA-security of the SKE scheme.) We rely here on the fact that, in both worlds, the simulator emulates the ZK proof using $\mathcal{S}_{\mathsf{zk}}$ as opposed to executing the real proof. In particular, the ZK simulator does not need the parties' private inputs for its simulation, and hence is not affected by the aforementioned change in the inputs.

Hybrid $\mathbf{H}\textsc{yb}^3_{\mathcal{A}(z^j)}(x_1^j, x_2'^j, \lambda)$: We modify the previous hybrid world, by having $\mathcal{S}_3^j$ provide an actual ZK proof that an input is re-used with respect to the state Σ' declared by the server, or to send an empty string ε if this is not case. The zero-knowledge property of the proof system automatically guarantees that the view generated using the real proof and the simulator $\mathcal{S}_{\mathsf{zk}}$ are computationally indistinguishable which in turn implies the computational indistinguishability of the output of the current hybrid experiment and the last. Thus, we have

$$\textsc{Hyb}^2_{f,\mathcal{A}(z^j),i}(x_1^j, x_2'^j, \lambda)_j \equiv_c \textsc{Hyb}^3_{f,\mathcal{A}(z^j),i}(x_1^j, x_2'^j, \lambda)_j.$$

To conclude the proof, it suffices to note that $\textsc{Hyb}^3_{f,\mathcal{A}(z^j),i}(x_1^j, x_2'^j, \lambda)_j$ exactly equals the output distribution of $\mathcal{A}^j$ in the real world, thus proving that $\widehat{\pi}_f \leftarrow \Psi_{\mathsf{rr}}(\pi_f, \mathscr{r}, \ell)$ is a secure (stateless) $\mathsf{rate\text{-}revealing}$ $\mathscr{r}$-limited protocol for function f. □

7 Rate-Limited OPE

Hazay and Lindell [HL09] design an efficient two-party protocol for oblivious polynomial evaluation (OPE) with security against malicious adversaries. In an OPE protocol, the first party holds a value x while the second party holds a polynomial p of degree d. Their goal is to let the first party learn $p(x)$ without revealing anything else. The protocol takes advantage of an additively homomorphic encryption scheme (Paillier's encryption) and efficient ZK proofs of a few statements related to the encryption scheme. While the authors (only) prove security against malicious adversaries, we observe that, with a small modification, their construction is indeed a commit-first protocol for OPE as well.

First party's commitment. Consider an additively homomorphic encryption scheme $(\mathsf{G}, \mathsf{E}, \mathsf{D})$. The first few steps performed by the first party (the party holding the value x) are as follows: (i) it runs the key generation for the encryption scheme to generate a key pair $(pk, sk) \leftarrow \mathsf{G}(1^\lambda)$, accompanied by a ZK proof of knowledge of the secret key; (ii) then, it encrypts powers of x, i.e. $\mathsf{E}(pk, x), \mathsf{E}(pk, x^2), \ldots, \mathsf{E}(pk, x^d)$, and sends the resulting ciphertexts along with a ZK proof of the validity of the ciphertexts to the other party.

We observe that sending $\mathsf{E}(pk, x)$ constitutes a commitment by the first party to its input t. This commitment scheme realizes the ideal functionality of the first phase in our definition of commit-first protocols. (Recall that this means the simulator can extract both the input and the randomness used to generate the commitment.) In particular, a careful inspection of the security proof of [HL09] reveals that the simulator can extract both x and the randomness used to encrypt it during the simulation. Extracting the randomness is possible since in Paillier's encryption scheme, given the secret key sk and a ciphertext c, one can recover both the randomness and the message.

Second party's commitment. The commitment of the second party to its input polynomial is slightly more subtle, and requires a small modification to the original design. In the first few steps, the second party does the following: (i) it runs the key generation to generate a key pair $(pk', sk') \leftarrow \mathsf{G}(1^\lambda)$, accompanied by a ZK proof of knowledge of the secret key; (ii) it computes $((\mathsf{E}(pk', q_1), \mathsf{E}(pk', p - q_1)), \ldots, (\mathsf{E}(pk', q_s), \mathsf{E}(pk', p - q_s))$ where q_i's are random polynomials of degree d for some security parameter s; (iii) it sends all the ciphertext pairs along with ZK proofs of the fact that the homomorphic addition of every pair encrypts the same polynomial (i.e., p), to the first party. We need to slightly modify this step to realize our ideal commitment functionality: For the first pair of ciphertexts, the second party will also include a ZK proof of validity of $(\mathsf{E}(pk', q_1), \mathsf{E}(pk', p - q_1))$.

The pair of ciphertexts $(\mathsf{E}(pk', q_1), \mathsf{E}(pk', p - q_1))$ and the accompanied ZK proof of their validity, constitute the commitment by the second party to its input polynomial p. Once again, we note that the simulator in the proof is able to extract q_1, p, and the randomness used in the two encryptions, due to the randomness recovering property of Paillier's encryption. The proof of security provided in [HL09] can be easily modified to show the commit-first property of the above-mentioned variant of their OPE construction.

Claim 7.1. *The modified oblivious polynomial evaluation protocol of [HL09] is a commit-first SFE with security against malicious adversaries.*

In the following subsections, we explain how to derive rate-limited OPE protocols from the scheme of [HL09], by giving concrete instantiation of our compilers from Section 5 and 6. In order to simplify the exposition, we consider the case of rate-revealing OPE first.

7.1 Rate-Revealing OPE

Consider our rate-revealing compiler from Section 5.2. A proof of repeated-input, here, is equivalent to proving a statement for the following language:

$$L^{\text{ope}}(n) = \left\{ \begin{array}{l} (pk, \hat{c}, c_1, \ldots, c_n) : \exists \lambda, r \text{ s.t. } (pk, sk) \leftarrow \mathsf{G}(1^\lambda; r) \text{ and} \\ (\mathsf{D}(sk, \hat{c}) = \mathsf{D}(sk, c_1) \vee \ldots \vee \mathsf{D}(sk, \hat{c}) = \mathsf{D}(sk, c_n)) \end{array} \right\},$$

where the ciphertexts $c_1, \ldots, c_n$ are encryptions of the inputs for n previous executions of the OPE protocol. The ciphertext $\hat{c}$ is the encryption of the input for the current execution.

Given language L^{ope}, the idea is to have the prover compute $\mathsf{E}(pk, (\hat{m} - m_1) \cdot \ldots \cdot (\hat{m} - m_n))$, prove correctness of this computation, and show that the final ciphertext is an encryption of zero. Consider the following languages:

$$\begin{array}{rcl} L^{\text{zero}} & = & \{(pk, c) : \exists r \text{ s.t. } c = \mathsf{E}(pk, 0; r)\}, \\ L^{\text{mult}} & = & \left\{ \begin{array}{l} (pk, c', c'', c) : \exists (m', m'', r', r'', r) \text{ s.t. } c' = \mathsf{E}(pk, m'; r') \wedge \\ \wedge c'' = \mathsf{E}(pk, m''; r'') \wedge c = \mathsf{E}(pk, m' \cdot m''; r)). \end{array} \right\} \end{array}$$

Using the protocols in [DJ01], a proof for L^{mult} requires 15 exponentiations and a proof for L^{zero} requires 8 exponentiations.

Given proof systems for the above languages, our proof π_{ope} can be constructed as follows.

Protocol π_{ope} (ZK proof for $L^{\text{ope}}(n)$)

- **Joint statement:** pk and $(\hat{c}, c_1, \ldots, c_n)$
- **Auxiliary inputs for the prover:** sk
- **Execution steps:**
  1. The prover sets $e_1 := \hat{c} -_h c_1$. Note that $\hat{c}$ and c_1 are encryptions obtained from an additive homomorphic encryption scheme, and $-_h$ denotes homomorphic subtraction. Therefore, the prover can derive the difference between the corresponding plaintexts in encrypted form.
  2. For $i = 2 \ldots n$, the prover
     (a) computes $d_i := \hat{c} -_h c_i$;
     (b) computes $e_i = \mathsf{E}(pk, \mathsf{D}(sk, e_{i-1}) \cdot \mathsf{D}(sk, d_i))$;
     (c) proves that $(e_{i-1}, d_i, e_i) \in L^{\text{mult}}$.
  3. The prover proves that $e_n \in L^{\text{zero}}$.

Proposition 1. *Assume π_{mult} (resp. π_{zero}) implements a ZK argument for languages L^{mult} (resp. L^{zero}). Then π_{ope} is a ZK argument for $L^{\text{ope}}(n)$.*

Proof Sketch. We need to show completeness, soundness and zero-knowledge.
Completeness. Correctness is easily shown. If there exists an index $i \in [n]$ such that $\mathsf{D}(sk, \hat{c}) = \mathsf{D}(sk, c_i)$, then d_i is an encryption of 0. Since in step (2c) the prover shows, using π_{mult}, that d_i is a divisor of e_i and e_{i-1}, this yields essentially that d_i is a factor of e_n. Thus, by correctness of π_{zero} the prover provides a valid proof.
Soundness. The prover is not able to proof a wrong statement but with negligible probability. In particular, if $\hat{c}$ encrypts a value different from all other plaintexts but e_n is indeed an encryption of 0, then it must have given a valid proof for a wrong statement for L^{mult}. That is, one of

the divisors d_i's (or e_1) encrypts 0. This contradicts soundness of π_{mult}. On the other hand, the prover needs to provide a valid proof for $e_n \in L^{\text{zero}}$, which is also computationally hard since π_{zero} is computationally sound by assumption.
Zero-Knowledge. The prover communicates with the verifier only when invoking the ZK protocols π_{mult} and π_{zero}. Since both protocols satisfy the zero-knowledge properties, the sequential execution of both protocols preserves zero-knowledge. □

7.2 Rate-Hiding OPE

Next, we explain how to derive a rate-hiding rate-limited OPE protocol from the scheme of [HL09], by giving a concrete instantiation of our compiler from Section 5.1. Note that besides a standard proof of the statement "a ciphertext is a valid encryption of bit b", in our rate-hiding compiler we also need proofs of membership in the following two languages:

$$L^{\text{rate}}_{\mathscr{r}}(n) = \left\{ \begin{array}{l} (pk, c_1, \ldots, c_n) : \ \exists \lambda, r \text{ s.t. } (pk, sk) \leftarrow \mathsf{G}(1^\lambda; r) \text{ and} \\ \sum_{1 \le i \le n} \mathsf{D}(sk, c_i) < \mathscr{r} \end{array} \right\}$$

and the complement of the $L^{\text{ope}}(n)$ language (see Section 7.1), i.e.:

$$L^{\overline{\text{ope}}}(n) = \left\{ \begin{array}{l} (pk, \hat{c}, c_1, \ldots, c_n) : \ \exists \lambda, r \text{ s.t. } (pk, sk) \leftarrow \mathsf{G}(1^\lambda; r) \text{ and} \\ (\mathsf{D}(sk, \hat{c}) \neq \mathsf{D}(sk, c_1) \wedge \ldots \wedge \mathsf{D}(sk, \hat{c}) \neq \mathsf{D}(sk, c_n)) \end{array} \right\}.$$

Given appropriate proofs for these languages, one can apply the techniques of [CDS94], to efficiently construct a proof for any conjunctive and/or disjunctive formula over statements proved in the components. As a result, we only need to show proofs for the above two languages.

We first show that in our case, proof of membership in $L^{\text{rate}}_{\mathscr{r}}$ can in fact be reduced to a proof for the language $L^{\overline{\text{ope}}}(n)$, and then describe a proof for the latter. To observe why this is the case, note that in our rate-hiding compiler c_i's are encryptions of 0 and 1. Thus, we compute the following n ciphertexts $c'_i = \sum_{1 \le j \le i} c_j$, where the sum represent the additive homomorphic operation. It is easy to see that there is an encryption of t among the c'_is *if and only if* there are at least t encryptions of 1 among the c_is. As a result, $L^{\text{rate}}_{\mathscr{r}}(n)$ for $(pk, c_1, \ldots, c_n)$ reduces to $L^{\overline{\text{ope}}}(n)$ for $(pk, c'_1, \ldots, c'_n)$.

It remains to show a proof for $L^{\overline{\text{ope}}}(n)$. The proof strategy is the same as the complement language discussed above until the last step. In the last step, instead of proving that the resulting ciphertext is encryption of 0, we need to prove that it is encryption of a non-zero. While it is possible to show such a statement using "range proofs" [CCS08], we show a direct and more efficient technique for proving this statement for the language

$$L^{\overline{\text{zero}}} = \{(pk, c) : \ \exists r, m \neq 0 \text{ s.t. } c = \mathsf{E}(pk, m; r)\}.$$

The idea is to multiply the underlying plaintext with a random value in the message domain (both parties contribute to the random value to avoid cheating). If the plaintext was a 0, so is the product, but if not, the result is non-zero with high probability. Furthermore, revealing the product does not reveal any information about the original plaintext (hence zero-knowledge).

Protocol $\pi_{\overline{\text{zero}}}$ (ZK proof for $L^{\overline{\text{zero}}}$)

- **Joint statement:** (pk, c)
- **Auxiliary inputs for the prover:** sk
- **Execution steps:**
  1. The prover generates a uniformly random message r_p from the message space and sends $c_p = \mathsf{E}(pk, r_p; r)$ to the verifier along with a standard proof of knowledge of message and randomness.

2. The verifier generates a uniformly random message r_v from the message space and sends $r_v, r', c_v = \mathsf{E}(pk, r_v; r')$ to the prover.
3. The prover lets $c_1 = c_v +_h c_p$, and $c_2 = \mathsf{E}(pk, (r_v + r_p)m; r'')$, and runs π_{mult} for the tuple (pk, c, c_1, c_2). The prover also reveals $(r_v + r_p)m$ and r''.
4. The verifier accepts if $(r_v + r_p)m$ is non-zero and the ciphertext c_2 was generated honestly.

The proof that the above protocol is indeed a ZK argument system follows along the lines of the proof of Proposition 1, and is therefore omitted.

8 Conclusions and Open Problems

We have introduced the concept of RL-SFE as a useful tool for enforcing secure metering of the number of distinct inputs used by the players in multiple executions of SFE protocols. In particular, we defined three flavors of RL-SFE and designed three generic compilers yielding RL-SFE protocols (one for each flavor of RL-SFE). Our compilers require each player to keep a state whose size grows linearly with the number of protocol executions. Motivated by this limitation, we have also shown how our compilers can be made stateless in the client-sever scenario, by having the client outsource its state at the server (in an authenticated and verifiable manner).

Our compilers for RL-SFE rely on so-called CF-SFE protocols, a concept that we formalized along the way and that we believe to be of independent interest. CF-SFE is a natural abstraction to divide the design of SFE protocols with malicious security into two stages: (i) A first stage where each party commits to its input; (ii) A second stage where the function to be computed is evaluated on the inputs the player committed to at the end of the first state. Whilst almost all SFE protocols with malicious security follow this blueprint, there are some exceptions, e.g. the protocol of [AMP04] for computing the k-th ranked element. It is an interesting open question how RL-SFE for such protocols can be achieved.

The present work focuses on (two-party) RL-SFE in the setting of *sequential* composition under static corruptions, where each protocol execution starts after the previous execution terminates, and moreover the identity of the corrupted player is fixed before the first execution begins. Extending our framework to the setting of adaptive security and/or concurrent and universally composable security [Can01] is left as an interesting direction for future research.

Finally, other natural extensions of our work would be to generalize RL-SFE to the multi-party setting, and to reduce the communication complexity of our protocols.

Acknowledgments

Work partially done while the first author was a PhD student at TU Darmstadt, and the last author was a postdoc at Aarhus University. We are grateful to Ivan Damgård and to the anonymous reviewers of PKC 2013 for the useful feedback provided on earlier versions of this paper.

References

[ADDV15] Giuseppe Ateniese, Özgür Dagdelen, Ivan Damgård, and Daniele Venturi. Entangled encodings and data entanglement. In *SCC@ASIACCS*, pages 3–12, 2015.

[ADDV16] Giuseppe Ateniese, Özgür Dagdelen, Ivan Damgård, and Daniele Venturi. Entangled cloud storage. *Future Generation Comp. Syst. (Special Issue on Cloud Cryptography)*, 2016.

[AMP04] Gagan Aggarwal, Nina Mishra, and Benny Pinkas. Secure computation of the kth-ranked element. In *EUROCRYPT*, pages 40–55, 2004.

[BCD$^+$09] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In *FC*, pages 325–343, 2009.

[BFK$^+$09] Mauro Barni, Pierluigi Failla, Vladimir Kolesnikov, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Secure evaluation of private linear branching programs with medical applications. In *ESORICS*, pages 424–439, 2009.

[BG89] Donald Beaver and Shafi Goldwasser. Multiparty computation with faulty majority. In *CRYPTO*, pages 589–590, 1989.

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS*, pages 192–206, 2008.

[BNO08] Amos Beimel, Kobbi Nissim, and Eran Omri. Distributed private data analysis: Simultaneously solving how and what. In *CRYPTO*, pages 451–468, 2008.

[BTW12] Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis. In *FC*, pages 57–64, 2012.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *IEEE FOCS*, pages 136–145, 2001.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols. In *ACM STOC*, pages 11–19, 1988.

[CCS08] Jan Camenisch, Rafik Chaabouni, and Abhi Shelat. Efficient protocols for set membership and range proofs. In *ASIACRYPT*, pages 234–252, 2008.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, pages 174–187, 1994.

[CKT10] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *ASIACRYPT*, pages 213–231, 2010.

[CS03] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO*, pages 126–144, 2003.

[CvdGT95] Claude Crépeau, Jeroen van de Graaf, and Alain Tapp. Committed oblivious transfer and private multi-party computation. In *CRYPTO*, pages 110–123, 1995.

[DGKN09] Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous multiparty computation: Theory and implementation. In *PKC*, pages 160–179, 2009.

[DJ01] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *PKC*, pages 119–136, 2001.

[DMV13] Özgür Dagdelen, Payman Mohassel, and Daniele Venturi. Rate-limited secure function evaluation: Definitions and constructions. In *PKC*, pages 461–478, 2013.

[DV14] Özgür Dagdelen and Daniele Venturi. A multi-party protocol for privacy-preserving cooperative linear systems of equations. In *BalkanCryptSec*, pages 161–172, 2014.

[FHV13] Sebastian Faust, Carmit Hazay, and Daniele Venturi. Outsourced pattern matching. In *ICALP*, pages 545–556, 2013.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, pages 303–324, 2005.

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, pages 1–19, 2004.

[Gar04] Juan A. Garay. Efficient and universally composable committed oblivious transfer and applications. In *TCC*, pages 297–316, 2004.

[GHS10] Rosario Gennaro, Carmit Hazay, and Jeffrey S. Sorensen. Automata evaluation and text search protocols with simulation based security. Cryptology ePrint Archive, Report 2010/484, 2010. `http://eprint.iacr.org/`.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *ACM STOC*, pages 25–32, 1989.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *ACM STOC*, pages 218–229, 1987.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity for all languages in NP have zero-knowledge proof systems. *J. ACM*, 38(3):691–729, 1991.

[Gol09] Oded Goldreich. *Foundations of Cryptography: Vol. 2, Basic Applications.* Cambridge University Press, 2009.

[HEK12] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.

[HEKM11] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, 2011.

[HKS$^+$10] Wilko Henecka, Stefan Kögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. TASTY: tool for automating secure two-party computations. In *ACM CCS*, pages 451–462, 2010.

[HL08] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *TCC*, pages 155–175, 2008.

[HL09] Carmit Hazay and Yehuda Lindell. Efficient oblivious polynomial evaluation with simulation-based security. *IACR Cryptology ePrint Archive*, 2009:459, 2009.

[HN12] Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. *J. Cryptology*, 25(3):383–433, 2012.

[HT10] Carmit Hazay and Tomas Toft. Computationally secure pattern matching in the presence of malicious adversaries. In *ASIACRYPT*, pages 195–212, 2010.

[IP07] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In *TCC*, pages 575–594, 2007.

[JS07] Stanislaw Jarecki and Vitaly Shmatikov. Efficient two-party secure computation on committed inputs. In *EUROCRYPT*, pages 97–114, 2007.

[Lin08] Yehuda Lindell. Efficient fully-simulatable oblivious transfer. *Chicago J. Theor. Comput. Sci.*, 2008, 2008.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, pages 52–78, 2007.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.

[MF06] Payman Mohassel and Matthew K. Franklin. Efficiency tradeoffs for malicious two-party computation. In *PKC*, pages 458–473, 2006.

[MHKS12] Piotr Mardziel, Michael Hicks, Jonathan Katz, and Mudhakar Srivatsa. Knowledge-oriented secure multiparty computation. In *PLAS*, page 2, 2012.

[MMP$^+$10] Andrew McGregor, Ilya Mironov, Toniann Pitassi, Omer Reingold, Kunal Talwar, and Salil P. Vadhan. The limits of two-party differential privacy. In *IEEE FOCS*, pages 81–90, 2010.

[MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay—Secure two-party computation system. In *USENIX Security Symposium*, pages 287–302, 2004.

[OK04] Wakaha Ogata and Kaoru Kurosawa. Oblivious keyword search. *J. Complexity*, 20(2-3):356–371, 2004.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[Ped92] Torben Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1992.

[TPKC07] Juan Ramón Troncoso-Pastoriza, Stefan Katzenbeisser, and Mehmet Utku Celik. Privacy preserving error resilient DNA searching through oblivious automata. In *ACM CCS*, pages 519–528, 2007.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *IEEE FOCS*, pages 160–164, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *IEEE FOCS*, pages 162–167, 1986.