LR-OT: Leakage-Resilient Oblivious Transfer

Francesco Berti, Carmit Hazay, Itamar Levi

Bar-Ilan University, Ramat-Gan, Israel
{francesco.berti, Carmit.Hazay, itamar.levi}@biu.ac.il

Abstract. Oblivious Transfer (OT) is a fundamental cryptographic primitive, becoming a crucial component of a practical secure protocol. OT is typically implemented in software, and one way to accelerate its running time is by using hardware implementations. However, such implementations are vulnerable to side-channel attacks (SCAs). On the other hand, protecting interactive protocols against SCA is highly challenging because of their longer secrets (which include inputs and randomness), more complicated design, and running multiple instances. Consequently, there are no truly practical leakage-resistant OT protocols yet.
In this paper, we introduce two tailored indistinguishability-based security definitions for leakage-resilient OT, focusing on protecting the sender's state. Second, we propose a practical semi-honest secure OT protocol that achieves these security levels while minimizing the assumptions on the protocol's building blocks and the use of a secret state. Finally, we extend our protocol to support sequential composition and explore efficiency-security tradeoffs.

1 Introduction

Oblivious Transfer (OT) is a fundamental cryptographic primitive that, in its simplest form, allows a sender to transmit a set of two messages, (m_0, m_1), to a receiver that receives only one, m_r, according to its selection bit r. The sender's goal is to hide the message that the receiver should not receive, while the receiver's goal is to hide r [47, 24]. Its security has been extensively studied in the presence of semi-honest corrupted adversaries (when the adversary follows the protocol's instructions but tries to violate the honest party's privacy), and malicious adversaries (when the adversary arbitrarily deviates from the protocol's execution) [44, 29]. The security of OT as an interactive protocol is typically formalized in the simulation-based paradigm, where the adversary's view can only be simulated from its inputs and outputs [29, 25].

The importance of OT lies in the fact that it is complete for secure computation [37, 57, 26, 32]. It also has some significant advantages, such as preprocessing [37, 3] (i.e., it can be precomputed in an offline stage before the actual inputs are known [37, 4]), being amortizable (i.e., we can *extend* OTs by using a few expensive OTs to generate a large number of cheap OTs, significantly reducing the overall cost [37, 31]), and being symmetric [56] (i.e., the roles of sender and receiver can be switched). On the other hand, it is impossible to build an OT scheme only from symmetric primitives [30, 4]. As such, OT has been studied under a variety of hardness assumptions, and its efficiency has been extensively explored with respect to both semi-honest and malicious attacks [14, 19, 48, 50].

When it comes to practical constructions, most OT implementations have been software-based, while more and more lower-level primitives implementations have been shifted toward hardware, making them faster; see, for instance, [21, 6]. The downside of such implementations is that they can more easily leak sensitive information using *side-channel attacks (SCA)*. Namely, an adversary can measure physical quantities such as time, power consumed, and electromagnetic radiation while running a computation. Using these physical quantities, it can extract information beyond the intended output [38, 39, 46]. In addition, it is possible to leak secret information previously stored in the memory even after the machine has been switched off via the *cold boot attacks* [28]. Such attacks are increasingly threatening due to the significant growth of IoTs, connected and physically accessible devices [40].

From a theoretical standpoint, SCAs are formalized using two classes of attacks [35]: 1) *memory leakage*, where the adversary obtains a bounded-output leakage function applied to the secret, e.g., [1]. 2) *computation leakage*, where each time a computational step involves a secret, a measurement of that secret can be made, also known by the "Only Computation Leaks" (OCL) paradigm [42]. The approach also considers leakage from intermediate ephemeral values generated during computation.

Several types of countermeasures have been considered against such attacks, working either at the hardware level to reduce the side-channel information or at the implementation level, exploiting data randomization (e.g., masking) or shuffling the order of operations. Since all these countermeasures increase the overheads, a complementary line of leakage-resistant cryptography works at the design-level (specifying how to use each key, how many times, etc.), in an attempt to design cryptographic primitives that are inherently more secure against SCA [23, 42, 59, 45, 2, 9] by limiting the number of times a secret is used. See [35] for additional citations.

However, when trying to formalize security and protect more involved interactive objects, this task becomes more challenging due to 1) longer secrets, 2) more complicated design involving multiple building blocks, and 3) reusability of secrets between different building blocks or multiple instances. As such, fewer examples of concrete leakage-resilient secure protocols are at the design level.

Relevant to our work, in [11], the authors introduced a new security model for secure computation where the adversary's view and the bounded leakage are simulatable by only accessing the adversary's input and output. They also built protocols achieving this security definition, using expensive primitives such as non-committing encryption schemes that allow the generation of equivocal ciphertexts. In another paper [15], the authors introduced an alternative security definition where the adversary's view and its unbounded leakage should be simulatable, assuming that each input is encoded leak-free. Even assuming leak-free pre-processing, secure constructions that consider leakage require the use of expensive primitives, such as fully homomorphic encryption.

Additional works provide protocols for specific functionalities, such as zero knowledge and coin tossing [35] with bounded leakage, and oblivious transfer [51], that provides a leaky OT protocol based on DDH. This protocol aims to protect the receiver's input, which is encrypted using a leak-free encryption phase but does not consider any countermeasures for the sender's secret state. We point out that there have also been attempts to secure cryptographic protocols at the so-called hardware level, protecting the circuit; see, for example, [13].

As it stands today, no practical solutions exist to protect the sender's state without using expensive primitives. Moreover, simulation-based security definitions may be challenging to use as it is not clear how to simulate leakage attacks considered in practice, such as leakage measurements of power consumption [41, 8], leaving a gap for the construction of practical, efficient OT protocols.

In this paper, we explore the security of oblivious transfer against SCA via two indistinguishability-based security definitions and provide several practical constructions that minimize the use of public-key primitives. With the understanding that protecting the receiver's single bit input is harder, we focus on protecting the sender's state.

1.1 Our Contribution

We describe our contributions as follows:

1. We give two indistinguishability-based security definitions tailored to OT in the presence of leakage when the goal is to protect the sender's privacy. Our definitions capture the following two scenarios: **(1)**, "intercepting side-channel adversaries" where an outsider eavesdrops on the protocol's transcript and learns the sender's associated computation leakage, and **(2)** "corrupted side-channel receivers" where the receiver is (semi-honestly) corrupted, and the adversary additionally learns the sender's associated computation leakage.
2. We design several OT protocols that satisfy semi-honest simulation-based security and the two definitions above. As we show, our protocols make minimal assumptions and minimally use the inputs and ephemeral secrets.
3. We extend the previous protocol to multiple instances to support a sequential composition. In this composition, the security of any instance relies on the security of the previous instance. We also discuss variants of our protocols, trading additional security requirements for efficiency.

1.2 Technical Overview

Modelling security in the presence of leakage. Our security holds in the OCL paradigm. In this model, *only computation leaks information* (i.e., information can be obtained about the bits involved in a computation, and it depends on the computation being performed) and *information leakage is local* (i.e., the information leaked by a computation depends only on the bits accessed by that computation, and it is independent of what has happened before and what will happen after [42]). This model makes security easier to prove.

Applying the OCL model to the case of interactive protocols, we divide a party's execution of the protocol into rounds and sub-computations. This allows us to specify which part of the inputs and intermediate values are touched for each sub-computation. We represent the leakage of a sub-computation obtained by an adversary by a leakage function, which takes as input the input of that computation as in [42]. Thus, the leakage of a party's protocol execution is given by concatenating all the leakage functions of the sub-computations performed by that party. Modeling security in the presence of leakage meaningfully requires restricting the informativeness of the leakage [59, 45, 9]. I.e., for tractability, we restrict the class of leakage functions to model the limited information an adversary realistically obtains via side-channel attacks. We apply these restrictions meaningfully to each building block in our construction. For instance, with a block cipher, we might assume the leakage does not fully reveal the secret states, assuming (the common) Hamming-weight leakage model. Ideally, such assumptions should undergo rigorous testing in certified evaluation LABs to validate them, concluding in statistical distances from assumed distributions.

Our security definitions for OT. Our definitions protect the sender's inputs, m_0 and m_1, in the presence of leakage using indistinguishability-based games. Namely, the adversary provides the sender with two sets of inputs and should not be able to distinguish which set the sender is using.

We consider two types of adversaries: 1) *intercepting side-channel adversaries* where the adversary can only see the transcript of the protocol (i.e., the messages exchanged by the parties) and, in addition, obtains leakage from the sender's secret state; 2) *corrupted side-channel receivers* where the adversary is stronger and can also corrupt the receiver. In both cases, the adversary should not be able to tell which sets of inputs the sender is using, even in the presence of leakage. We leave open the problem of using our security definitions in the general context of leakage-resistant interactive objects and secure computations.

$\pi^{\mathsf{LR\text{-}OT}}$*, a single instance leakage-resistant protocol.* A natural paradigm for designing leakage-resilient OT is to use leakage-resistant building blocks. Thus, we start with the simplest leakage-resilient object: a leakage-resistant symmetric encryption scheme (intuitively, the more complex a primitive is, the harder it is to protect its implementation). That is, the parties share a symmetric key k (which can be obtained via a key-exchange protocol). Following that, S encrypts its inputs m_0 and m_1, obtaining $c_i = \mathsf{Enc}_k(m_i)$, and sends (c_0, c_1) using an OT protocol π^{OT}. By targeting π^{OT} with leakage, the adversary can only recover c_0 and c_1, which give no information about m_0 or m_1. Therefore, π^{OT} does not need additional protection against side-channel intercepting adversaries.

This, however, is insufficient against an adversary that corrupts the receiver R because such an adversary would know k. This implies that A can obtain information about $m_{\bar{r}}$ *via leakage* from $c_{\bar{r}}$.[1]. At a high level, to resolve this challenge, we use two layers of encryption schemes: first, to encrypt two random

[1] Recall that in the security definitions of encryption schemes with leakage, we assume that the adversary knows the challenge ciphertext but has at most some leakage in-

keys k_0 and k_1 (to get y_0 and y_1) using the key k, and then to use k_i to encrypt the messages m_i, i.e. $c_i = \mathsf{Enc}_{k_{i,E}}(m_i)$. The sender then uses π^{OT} with inputs (y_0, y_1). The idea is that for an adversary to learn anything about $c_{\bar{r}}$, it needs to guess $k_{\bar{r}}$, and to do that it needs to entirely learn $y_{\bar{r}}$ via leakage. Thus, to provide security against corrupted side-channel receivers, we need to use a leakage-resilient OT protocol. This may seem like a circular problem: to have a leakage-resistant $\pi^{\mathsf{LR\text{-}OT}}$, we need a leakage-resistant protocol, π^{OT}. Nevertheless, we require a weaker π^{OT} guarantee. Namely, instead of requiring the indistinguishability of its inputs, as we require for $\pi^{\mathsf{LR\text{-}OT}}$, we require unpredictability for a random input. That is, the corrupted receiver should fully guess $y_{\bar{r}}$ the sender's random input of π^{OT}, which R should not receive. It is an interesting open problem to explore whether we can eliminate the use of such an OT.

Note that unpredictability for a random input is a weaker assumption than the indistinguishability of two chosen inputs [42, 7]. Namely, to break the latter, it is enough to predict one bit of information about $y_{\bar{r}}$, while the former requires full recovery of $y_{\bar{r}}$ [22]. For example, if the implementation of π^{OT} leaks, the exact Hamming weight of $y_{\bar{r}}$, π^{OT} would not provide indistinguishability for selected inputs but would provide unpredictability for a random input.

According to our leakage model, we divide the sender's protocol execution into sub-computations (each involving a single building block, sampling of randomness, or an XOR computation) and identify which secret values can be leaked for each sub-computation. We require each block to be unpredictable in the presence of leakage, except for the encryption scheme, for which we require eavesdropping security with leakage [45]. However, it is not enough to bind the leakage functions for each block since different blocks may use the same secret where the adversary can combine their leakage to retrieve this secret (for example, in our protocol, the sender obtains the key k via the key-exchange protocol and uses it twice as a key to encrypt k_0 and k_1, so an adversary can obtain the left half of k via leakage of the key-exchange, and the right half from the encryption of k_0 and k_1. Although both leakages are not enough to break the unpredictability of these blocks, an adversary who obtains both can break our scheme). Thus, we also require that when different blocks use the same secret value, the composition of their leakages does not reveal secrets. To minimize the threat of a side-channel adversary exploiting combined leakages, we design our scheme so that each secret is used by at most two different primitives (plus possibly the random sampling and the XOR function), for more details see Sec. 4. We leave the problem of instantiating these blocks for future work.

Sequential leakage resilient OT. As for multiple instances, $\pi^{N\text{-}\mathsf{LR\text{-}OT}}$, we exploit the fact that in the previous $(i-1)^{\text{th}}$ instance, the sender has two keys, $k_{i-1,0}$, $k_{i-1,1}$, one of which is known to the receiver, $k_{i-1,r_{i-1}}$, where r_{i-1} is the receiver's input bit for the $(i-1)^{\text{th}}$ instance. To achieve sequential composition, we execute the first instance as in the $\pi^{\mathsf{LR\text{-}OT}}$ protocol. Moreover, we use the sender's keys

formation about the key, here the adversary knows the key and has some information about the challenge ciphertext.

for each iteration to encrypt the picked keys for the next one. This introduces four ciphertexts, $y_{i,00}, y_{i,10}, y_{i,01}, y_{i,11}$. Similarly to the single instance, the sender refreshes the keys $\{k_{i,j}\}$ with F to generate the keys $\{k_{i,j,E}\}$ which are used to encrypt the sender's input to this instance; moreover, the sender and the receiver perform an OT protocol where the sender's inputs are $((y_{i,00}, y_{i,10}), (y_{i,01}, y_{i,11}))$ and the receiver is r_i. We introduce additional variants and improvements in Sec 6, covering additional composition for multiple instances and more efficient alternatives based on stronger cryptographic building blocks.

To conclude, the area of leakage-resilient secure computation will continue to draw attention as the use of hardware becomes more essential for achieving practical implementations, as recently occurred for zero-knowledge proof systems.

2 Background

Notations. Let $[n] = \{1, \ldots, n\}$. We use $\perp$ to denote the empty string. Let $\{0,1\}^n$ be the set of all n-bit long strings and $\{0,1\}^*$ the set of all finite strings ($\perp \in \{0,1\}^*$). Let x be a bit. With $\bar{x}$ we denote the bit $x \oplus 1$. Let $\mathcal{X}$ be a set. With $\mathsf{dist}(\mathcal{X})$ we denote the set of the probabilistic distributions over the set $\mathcal{X}$. By $x \overset{\$}{\leftarrow} \mathcal{X}$ we denote that x is picked uniformly at random from the set $\mathcal{X}$.

2.1 Adversaries and Leakage

When we evaluate security, we define the adversary and its abilities [5, 45, 27]:

Definition 1. *A (q,t)- (black-box) adversary A^{O} is a probabilistic algorithm with oracle access to O that makes at most q queries and runs at most in time t.*

By $y \leftarrow \mathsf{A}^{\mathsf{O}_1,\ldots,\mathsf{O}_n}(x)$, we denote that the adversary A outputs y on input x, with oracle access to $\mathsf{O}_1, \ldots, \mathsf{O}_n$.

Leakage. In the real world, when an adversary interacts with an oracle, it can also obtain physical information about the oracle's computations, the so called *leakage* [38, 39, 46] is then exploited by side-channel attacks (SCA) to perform powerful attacks. We model leakage using the *leakage function* [2, 23].

Definition 2. *The* leakage *of an oracle O is represented by the function*

$$\mathsf{L}_{\mathsf{O}} : \mathcal{I}^{sec} \times \mathcal{I}^{pub} \times \mathcal{R} \to \{0,1\}^*,$$

where $\mathcal{I}^{pub}$ is the set of inputs chosen by the adversary, $\mathcal{I}^{sec}$ is the set of secret inputs (e.g., the secret key [2]), and $\mathcal{R}$ denotes the set from which the randomness used in the computation is selected.

[2] For example, if the oracle implements an encryption scheme $\mathsf{Enc}_k(m)$, and the adversary plays a CPA-game with leakage, the public input is m, chosen by the adversary, while the secret input is the key k. Thus, $\mathcal{I}^{sec} = \mathcal{K}$, the key-space of the encryption scheme, while $\mathcal{I}^{pub} = \mathcal{M}$, the message space.

Leaking oracles are denoted with the suffix L; for example, we denote the oracle O as OL if it leaks [45]. So, when an adversary queries the oracle O_k on input (x), it receives $(y, \mathsf{L_O}(k; x, R))$ where y is the output $\mathsf{O}_k(x)$, R is the randomness used in the computation. The function $\mathsf{L_O}$ models what a real adversary receives when performing a side-channel attack on the device performing the cryptographic computations. If the leakage function of the oracle always returns the empty string ϵ, we say that the oracle does not leak. An adversary which receives the leakage is called a *side-channel adversary* [27]:

Definition 3. *Let* L *be the leakage function of the oracle* O *(Def. 2). A* side-channel *adversary* A^{OL} *is an adversary which, when it interacts with an oracle* O*, receives the outputs and the leakage of* O.

Our syntax allows us to separate the process of obtaining the leakage via physical measurement, which we model by the leakage functions (abstracting its complexity), from the internal computation of the adversary [45]. Clearly, for a given secret k used by the oracle O, the more evaluations of $\mathsf{L_O}(k; \cdot, \cdot)$, denoted *traces* in the SCA community jargon, the easier it is for an adversary to retrieve k [45].

2.2 Leakage-Resilient Symmetric Cryptography

In this section, we introduce the two symmetric primitives we will use in our protocol: block-ciphers and encryption schemes.

Block-Ciphers. We start by introducing block-ciphers [36], which we will use to generate pseudo-random strings.

Definition 4. *A n-bit* block-cipher *(*BC*)* $\mathsf{F} : \mathcal{K} \times \{0,1\}^n \rightarrow \{0,1\}^n$ *is a family of permutations* $\mathsf{F}_k : \{0,1\}^n \rightarrow \{0,1\}^n$. $\mathsf{F} : \mathcal{K} \times \{0,1\}^n \rightarrow \{0,1\}^n$ *is a* (q,t,ϵ)-PRF *(*Pseudo Random Function*) if for any* (q,t)*-adversary* A

$$|\Pr[1 \leftarrow \mathsf{A}^{\mathsf{F}_k(\cdot)}] - \Pr[1 \leftarrow \mathsf{A}^{\mathsf{f}(\cdot)}] \leq \epsilon$$

where $k \xleftarrow{\$} \mathcal{K}$, $\mathsf{f} \xleftarrow{\$} \mathsf{FUNC}$ (FUNC *is the set of the functions* $\{0,1\}^n \rightarrow \{0,1\}^n$).

In some proofs, we will model F as an *ideal cipher* [17]. This means that F is seen as a family of $|\mathcal{K}|$ independent permutations (for the formal definition:

Definition 5 ([17]). *A block-cipher* $\mathsf{F} : \mathcal{K} \times \{0,1\}^n$ *is an* ideal cipher *if it has been chosen uniformly at random among all block-ciphers with the same input and output spaces.*

As in the random oracle model [36], $\mathsf{F}_k(x)$ can only be evaluated by making a query to the oracle F on the input (k, x), yielding $y = \mathsf{F}_k(x)$. Thus, if F has never been queried on input (k, x), then, $y = \mathsf{F}_k(x)$ is random.

We assume that whenever an adversary A interacts with an oracle O that needs to query the ideal cipher F, then, A can query F too.

Next, we consider the security of the ideal cipher F in the presence of leakage. In practice, an adversary attacks an actual implementation of F. To model the attack, we capture the leakage both in the key-generation, with the leakage function $\mathsf{L}^{gen}(\cdot)$, and in the actual computation of F, with $\mathsf{L}_\mathsf{F}(\cdot;\cdot)$. To prevent leakage functions of the computation $y = \mathsf{F}_k(x)$ from giving information about future (or past) computations, we assume that L_F has no access to F and we consider leakage functions of this form: $\mathsf{L}_\mathsf{F}(k;x) := (\mathsf{L}_\mathsf{F}^{in}(k;x), \mathsf{L}_\mathsf{F}^{out}(k;y))$, where $\mathsf{L}_\mathsf{F}^{in}$ leaks information about the input and the key k, and $\mathsf{L}_\mathsf{F}^{out}$ about the output and k [59]. Having no oracle access to F, $\mathsf{L}_\mathsf{F}^{in}(k;x)$ cannot compute y. This modeling captures concrete attacks on actual block cipher implementations [59, 9] that focus either on the first rounds (so that the leakage can be seen as a function of the input and the key) [58, 54, 12, 55, 40] or on the last rounds (so that the leakage can be seen as a function of the key and output) or both [18].

We consider an implementation of F to be secure if, given $\mathsf{L}^{gen}(k)$ and oracle access to $\mathsf{FL}_k(\cdot)$, the key cannot be guessed. We give A the ability to output a set $\mathcal{G}$ of q_G keys, and A wins if $k \in \mathcal{G}$) [9]. If the adversary cannot guess the key k, due to the ideal cipher model, F_k is a random permutation.

Definition 6. *Let* $\mathsf{F} : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ *be an ideal cipher and* L *be a leakage function. Let* k *be the key used by* F *and* $\mathsf{L}^{gen}(k)$ *be the leakage of its generation, we say that* F *is* $(q, q_G, q_\mathsf{F}, t, \epsilon)$-unpredictable with leakage L and key-generation leakage $\mathsf{L}^{gen}(\cdot)$ *(*q-upL*) if for any* (q, q_F, t)*-adversary* A,

$$\Pr[k \in \mathcal{G} \mid |\mathcal{G}| \leq q_G,\ \mathcal{G} \leftarrow \mathsf{A}^{\mathsf{FL}_k(\cdot), \mathsf{F}.(\cdot)}(\mathsf{L}^{gen}(k)),\ k \xleftarrow{\$} \mathcal{K}] \leq \epsilon,$$

where the adversary is granted q_F *queries to the ideal cipher* F *(it also chooses the key in these queries) and* q *queries to* FL_k*. For the latter queries, on input* x*, the adversary gets* $y = \mathsf{F}_k(x)$*, and the leakage* $(\mathsf{L}_\mathsf{F}^{in}(k;x), \mathsf{L}_\mathsf{F}^{out}(k;y))$*.*

Encryption schemes. We use *encryption schemes* to send messages privately. They consist of three algorithms $\mathsf{ENC} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, to generate the key k, to encrypt a message m with the key k, and to decrypt a ciphertext c with the key, respectively. We use *correct* encryption schemes, i.e., for all possible k and m, $m = \mathsf{Dec}_k(\mathsf{Enc}_k(m))$. We now give the formal syntax.

Definition 7. *An* encryption scheme *(*ENC*) scheme is a triple of algorithms* $\mathsf{ENC} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *where*

- *The key-generation algorithm* Gen *picks a key uniformly at random from the set of keys,* $\mathcal{K}_\mathsf{ENC}$*.*
- *The encryption algorithm* Enc *is a probabilistic algorithm that takes as input a key* $k \in \mathcal{K}_\mathsf{ENC}$*, and a message* $m \in \mathcal{M}$*, and outputs a ciphertext* $c \in \mathcal{C}$*. We denote this with* $c \leftarrow \mathsf{Enc}_k(m)$*.*
- *The decryption algorithm* Dec *is a deterministic algorithm that takes as input a key* $k \in \mathcal{K}_\mathsf{ENC}$*, and a ciphertext* $c \in \mathcal{C}$*, and outputs a message* $m \in \mathcal{M}$ *or* $\perp$ *("invalid"). We denote this with* $\perp / m = \mathsf{Dec}_k(c)$*.*

We need correctness*, so* $\forall (k, m) \in \mathcal{K}_\mathsf{ENC} \times \mathcal{M}$*,* $m = \mathsf{Dec}_k(\mathsf{Enc}_k(m))$*.*

We use probabilistic encryption schemes (for other syntax, see [43]).

We require that encryption schemes do not reveal any information about the message they encrypt even in the presence of leakage. To do this, we use the CPAL-security definition [45], which based on the well-known *Chosen-Plaintext Attacks*-security (CPA) [36], and EavL (Eavesdropper security).

Definition 8. *Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme. Let L be its leakage function. The encryption scheme Π is $(q_{\mathsf{L}}, q_E, t, \epsilon)$-*$\mathsf{CPAL}$-secure ($\mathsf{CPA}$ with leakage) *in the presence of* L*, if for any (q_E, t)-adversary $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$*

$$\Big| \Pr[b = b' \mid b' \leftarrow \mathsf{A}_2^{\mathsf{EncL}_k(\cdot)}(c^*, \ell^*, \mathsf{st}),\ (c^*, \ell^*) = \mathsf{EncL}_k(m_b),$$

$$(\mathsf{st}, m_0, m_1) \leftarrow \mathsf{A}_1^{\mathsf{EncL}_k(\cdot)} \ s.t.\ |m_0| = |m_1|, b \overset{\$}{\leftarrow} \{0,1\}] - \frac{1}{2} \Big| \leq \epsilon$$

where $k \leftarrow \mathsf{Gen}$, A_1 is a $(q_{1,E}, t_1)$-adversary, A_2 is a $(q_{2,E}, t_2)$-adversary, with $q_{1,E} + q_{2,E} \leq q_E$ and $t_1 + t_2 \leq t$. If $q_E = 0$, we say that Π is (q_E, t, ϵ)-EavL-secure (eavesdropper security with leakage) *in the presence of leakage* L.

The black-box definitions (CPA and Eav) can be obtained from CPAL and EavL respectively, by simply removing the leakage. Noteworthy, we have chosen to use CPAL and not $\mathsf{IND\text{-}CPAL}$ [2, 20] because in the latter definition A does not get ℓ^*, the leakage of the challenge ciphertext c^* computation. Namely, we need to work with CPAL since we have to consider this leakage in our proof.

2.3 Secure Two-Party Computation

In this section, we discuss secure two-party protocols, following the terminology of [29]. We consider *semi-honest* adversaries, that is, adversaries that control one of the parties. This party follows the protocol exactly, but wants to learn more information than its intended output of the protocol. Furthermore, we assume that the adversary is *static*, i.e., it chooses at the beginning of the computation of the party it will corrupt (i.e., take control of).

A *two-party protocol problem* can be defined by specifying a random process that maps pairs of inputs to pairs of outputs, known as a *functionality* $\mathcal{F}$. The input and output spaces of the ith party are denoted as $\mathcal{I}_i$ and $\mathcal{O}_i$ respectively, and $\mathcal{F}$ can be written as $\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2) : \mathcal{I}_1 \times \mathcal{I}_2 \rightarrow \mathcal{O}_1 \times \mathcal{O}_2$.

Denoting the input of the first party, P_1, by x^1, and the input of the second party, P_2, by x^2, P_1 obtains $\mathcal{F}_1(x^1, x^2)$, while P_2 obtains $\mathcal{F}_2(x^1, x^2)$. So we can also define the functionality using $(x^1, x^2) \mapsto (\mathcal{F}_1(x^1, x^2), \mathcal{F}_2(x^1, x^2))$. When the functionality is probabilistic, we denote it as $\mathcal{F}(x^1, x^2; R)$, where R is a uniformly chosen random tape used in the computation of $\mathcal{F}$.

To define correctness and security we need to introduce the *computational indistinguishability* of distributions. We say that two distributions $\mathcal{A} = \{A(x, n)\}_{x \in \{0,1\}^*, n \in \mathbb{N}}$ and $\mathcal{B} = \{B(x, n)\}_{x \in \{0,1\}^*, n \in \mathbb{N}}$, are *computationally indistinguishable*, which we denote by $\mathcal{A} \overset{c}{\underset{\epsilon}{\equiv}} \mathcal{B}$ if for every non-uniform polynomial-time algorithm A, there exists a negligible function $\epsilon(\cdot)$ s.t. for all $x \in \{0,1\}^*$, $n \in \mathbb{N}$,

$$|\Pr[1 \leftarrow \mathsf{A}(A(x, n))] - \Pr[1 \leftarrow \mathsf{A}(B(x, n))]| \leq \epsilon(n).$$

When A has running time bounded by $t(n)$, we say that $\mathcal{A} \overset{c}{\underset{t,\epsilon}{\equiv}} \mathcal{B}$.

Let $\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2)$ be a probabilistic polynomial-time functionality and π be a two-party protocol for computing $\mathcal{F}$. The view of the party P_j during the execution of π with input (x^1, x^2) and security parameter 1^κ is denoted by $\mathsf{view}_j^\pi(x^1, x^2, 1^\kappa)$ and is equal to $(x^j, R^j, \mathsf{trans})$, where R^j is the random tape used by P_j and trans is the transcript of the execution of the protocol. The output of party P_j is denoted by $\mathsf{outpt}_j^{\pi}(x^1, x^2, 1^{\kappa})$, and the *protocol output* is the joint output of the parties, that is $(\mathsf{outpt}_1^{\pi}(x^1, x^2, 1^{\kappa}), \mathsf{outpt}_2^{\pi}(x^1, x^2, 1^{\kappa}))$.

Definition 9. *We say that a protocol π is* correct *if*

$$\{\mathsf{outpt}^{\pi}(x^1, x^2, 1^{\kappa})\}_{x^1,x^2\in\{0,1\}^*,\kappa\in\mathbb{N}} \overset{c}{\equiv} \{\mathcal{F}(x^1, x^2)\}_{x^1,x^2\in\{0,1\}^*}.$$

The idea is that a secure protocol is one where the execution of the protocol does not give any party more information than that party can learn from its inputs and outputs. We formalise this using the *simulation* paradigm. Roughly, we want the party's view to be *simulatable* given its inputs and outputs. Formally,

Definition 10 ([29]). *Let $\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2)$ be a functionality. We say that π* securely computes $\mathcal{F}$ in the presence of static semi-honest adversaries *if there exist probabilistic polynomial-time algorithms Sim_1 and Sim_2 s.t.*

$$\{\mathsf{Sim}_1(1^{\kappa}, x^1, \mathcal{F}_1(x^1, x^2)), \mathcal{F}(x^1, x^2)\}_{x^1,x^2,1^{\kappa}} \overset{c}{\equiv} \{\mathsf{view}_1^{\pi}(x^1, x^2, 1^{\kappa}), \mathsf{outpt}^{\pi}(x^1, x^2, 1^{\kappa})\}_{x^1,x^2,1^{\kappa}}$$

$$\{\mathsf{Sim}_2(1^{\kappa}, x^2, \mathcal{F}_2(x^1, x^2)), \mathcal{F}(x^1, x^2)\}_{x^1,x^2,1^{\kappa}} \overset{c}{\equiv} \{\mathsf{view}_2^{\pi}(x^1, x^2, 1^{\kappa}), \mathsf{outpt}^{\pi}(x^1, x^2, 1^{\kappa})\}_{x^1,x^2,1^{\kappa}}$$

with κ in $\mathbb{N}$.

That is, the view of the parties can be simulated by an algorithm that has access *only to the input and the output* of the party.

2.4 Security Definition for Deterministic Functionalities

If the functionality $\mathcal{F}$ is deterministic, we have a simpler notion of security, because we can ask separately for the correctness of the protocol and that the simulator simulates the view in an indistinguishable way (see Def. 11). This definition is a simplified version of Def. 10 for 2-party protocols, when the functionality the protocol implements is deterministic.

Definition 11 ([29]). *Let $\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2)$ be a deterministic functionality. We say that π* securely computes $\mathcal{F}$ in the presence of static semi-honest adversaries *if π is correct (Def. 9) and there exist PPT algorithms Sim_1 and Sim_2 s.t.*

$$\{\mathsf{Sim}_1(1^{\kappa}, x^1, \mathcal{F}_1(x^1, x^2)\}_{x^1,x^2,1^{\kappa}} \overset{c}{\equiv} \{\mathsf{view}_1^{\pi}(x^1, x^2, 1^{\kappa})\}_{x^1,x^2,1^{\kappa}}$$

$$\{\mathsf{Sim}_2(1^{\kappa}, x^2, \mathcal{F}_2(x^1, x^2)\}_{x^1,x^2,1^{\kappa}} \overset{c}{\equiv} \{\mathsf{view}_2^{\pi}(x^1, x^2, 1^{\kappa})\}_{x^1,x^2,1^{\kappa}}$$

with κ in $\mathbb{N}$. If they are computationally indistinguishable except with probability ϵ, we say that π ϵ-securely computes *$\mathcal{F}$.*

2.5 Key-Exchange

This functionality aims to provide both parties with a random secret key.

Definition 12. *The* key-exchange *functionality is a two party functionality in which both parties agree to a random output of a certain length. That is,*

$$\mathcal{F}_{\mathsf{KE}} : (1^{\lambda}, 1^{\lambda}) \longmapsto (k, k) \text{ with } k \overset{\$}{\leftarrow} \{0,1\}^{\lambda}.$$

The security of a 2-party key-exchange protocol is covered by Def. 10.

Note that usually a key-exchange protocol is used only once to produce a given key. Thus, a side-channel adversary can only obtain a single *leakage* (trace in the jargon). I.e., such a trace is provided from an evaluation of $\mathsf{L}_{\pi^{\mathsf{KE}}}$. Therefore, owing to the fact that such leakage traces are noisy the adversary should not retrieve significant key material, i.e., contradicting unpredictability, especially if π^{KE} is lightly SCA-secured (E.g., constant time and lightly masked and/or shuffled implementation), since a *Single-Trace* SCA attack on asymmetric primitives are very challenging [33, 49, 34, 53]. We give the security definition for key-exchange protocols in the presence of leakage, Def. 20, Sec. 4.2.

2.6 Oblivious Transfer

The *1-out-of-2 oblivious transfer (*OT*) functionality* is a two-party functionality in which the sender, S, inputs two strings, m_0 and m_1, with $|m_0| = |m_1|$, while the receiver, R, inputs only one bit, r, and obtains m_r without passing any information about r to the sender and without receiving any information about $m_{\bar{r}}$ $(\bar{r} = r \oplus 1)$. That is, $\mathcal{F}_{\mathsf{OT}} : ((m_0, m_1), r) \longmapsto (\perp, m_r)$.

We use $m_{\bar{r}}$ to denote the message that the receiver does not learn. Def. 11 gives the security of a 2-party OT protocol since OT is a deterministic functionality. One of the goals of the paper is to provide security definitions for OT protocols in the presence of leakage; see Sec. 3.2, Def. 15, and Def. 16.

3 Security Definitions with Leakage

In this section, we present our security model. In Section 3.1, we will describe the leakage model. Then, in Section 3.2, we will provide the security definitions in the presence of leakage.

3.1 Computational Model and Leakage

Let π be a 2-party protocol, and let P_1 and P_2 be the parties executing the protocol. In this section we specify the different ingredients of a protocol and computation (i.e., inputs, states, messages, rounds and sub-computations), and then follow with defining their associated leakages.

Inputs, states and messages. We assume that both parties have many arrays of memory. Each array contains many cells, and each cell contains a finite string. We assume that the memory cells can only be written to once and that their contents can never be erased once they have been written to.

We assume that both parties have an array dedicated to their input. They also have an array dedicated to the intermediate results. This array, denoted as the *state*, is secret, and we denote by st^j the state of the party P_j. For each sub-protocol called, we assume that there is a sub-array of the state array dedicated to the intermediate computations of the execution of the sub-protocol, and an array for the inputs of the sub-protocol.

During the execution of a protocol, the parties exchange *messages* $\mathsf{msg}_1, \ldots, \mathsf{msg}_n$. Each message is a string. The transcript trans consists of all the messages exchanged between the parties to the protocol.

Setup sub-protocol. A protocol π may contain a preliminary sub-protocol $\pi.\mathsf{Setup}$ which takes as input the security parameter 1^κ and the randomness R^1_{set} and R^2_{set} of parties P_1 and P_2, respectively, and outputs the initial secret state of each party, denoted by st^1_0 and st^2_0, and the public parameters, pp (e.g. a group description and a generator). That is, $(\mathsf{st}^1_0, \mathsf{st}^2_0, \mathsf{pp}) \leftarrow \pi.\mathsf{Setup}(1^\kappa, R^1_{set}, R^2_{set})$. Looking ahead, since our protocol does not require an initial secret state (i.e., $\mathsf{st}^1_0 = \mathsf{st}^2_0 = \epsilon$), we can assume that all inputs, randomness, and outputs of this subprotocol are leaked. Namely, $\mathsf{L}_{set}(1^\kappa, R^1_{set}, R^2_{set}) = (\mathsf{pp}, R^1_{set}, R^2_{set})$.

Rounds. A protocol π proceeds through a sequence of n_π rounds of communication, with only one party speaking in each round. During each round, an outgoing message, msg_i for the i^{th} round, is computed by the *next message function* π_i for all $i = 1, \ldots, n_\pi$, taking the public parameters, the party's secret state and input, and the incoming message, with access to independent random values for each round. In addition, π_i produces a new secret state st_i, which may contain intermediate values obtained within the previous computations or incoming messages. According to our memory model, $\mathsf{st}^j_i = \mathsf{st}^j_{i-1} \| \mathsf{st}^{c,j}_i$ with $\mathsf{st}^{c,j}_i$ computed by the next round function (c denotes "computed"). Without loss of generality, we assume that P_1 speaks first. So, for odd i's, P_1 computes $(\mathsf{st}^1_i, \mathsf{msg}_i) \leftarrow \pi_i(\mathsf{pp}, \mathsf{st}^1_{i-1}, x^1, R^1_i, \mathsf{msg}_{i-1})$, where $\mathsf{st}^1_{i-1} := \mathsf{st}^1_{i-2}$, because in the $(i-1)^{\text{th}}$ round, P_1 was idle. (For $i = 1$, we set $\mathsf{msg}_0 = \perp$ since there is no msg_0). Then, P_1 sends msg_i to P_2. Moreover, in even rounds, P_2 computes $(\mathsf{st}^2_i, \mathsf{msg}_i) \leftarrow \pi_i(\mathsf{pp}, \mathsf{st}^2_{i-1}, x^2, R^2_i, \mathsf{msg}_{i-1})$, with $\mathsf{st}^2_{i-1} := \mathsf{st}^2_{i-2}$.

After completing n_π rounds, each party P_j produces its own output by computing $y^j \leftarrow \pi^j_{out}(\mathsf{pp}, \mathsf{st}^j_{n_\pi}, x^j, R^j_{out}, \mathsf{msg}_{n_\pi})$, via the *output functions* π^j_{out}, which produces the output outpt^π_j (as specified in Sec. 2.3).

Sub-computations decomposition. In order to handle leakage attacks, we decompose the next message function computation into smaller units of computation, which can be thought of as atomic building blocks. This formalization is inspired by the atomic model of [10] and captures cryptographic building blocks such as block ciphers or encryption schemes. In more detail, consider the next message function π_i of the ith round that is performed by P_j:

$$(\mathsf{st}^j_i, \mathsf{msg}_i) \leftarrow \pi_i(\mathsf{pp}, \mathsf{st}^j_{i-1}, x^j, R^j_i, \mathsf{msg}_{i-1}).$$

Next we will break this computation into a sequence of n_i sub-computations denoted by $\{\mathsf{Comp}_{i.i'}\}_{i' \in [n_i]}$ (the number of sub-computations can be different for each round). Each such component is considered as an atomic unit, with its own input and output state. The first sub-computation uses the secret state computed by the last sub-computation of the $(i-1)$th round, i.e., $\mathsf{st}^j_{i.0} := \mathsf{st}^j_{i-1}$. Then, each subsequent sub-computation updates the state accordingly. Without loss of generality, let us assume that each sub-computation uses its own independent random tape $R^j_{i.i'}$, and outputs $\mathsf{pmsg}_{i.i'}$, denoting a share of the ith message, msg_i, which can be empty. That is, for all $i' \in [n_i]$,

$$(\mathsf{st}^j_{i.i'}, \mathsf{pmsg}_{i.i'}) \leftarrow \mathsf{Comp}_{i.i'}(\mathsf{pp}, \mathsf{st}^j_{i.i'-1}, x^j, R^j_{i.i'}, \mathsf{msg}_{i-1}).$$

The output state of the last sub-computation, $\mathsf{st}^j_{i.n_i}$, is the output state of the ith round, st^j_i. The final outgoing message msg_i of the ith round is obtained by concatenating all the shares as follows, $\mathsf{msg}_i := \mathsf{pmsg}_{i.1}\| \dots \|\mathsf{pmsg}_{i.n_i}$. where a share[3] may be empty and the final outcome is captured by

$$\mathsf{pmsg}_{i.n_i} = g_i(\mathsf{pp}, \mathsf{st}^j_{i.n_i-1}, x^j, R^j_{i.n_i-1}, \mathsf{msg}_{i-1})$$

with g_i any function.

Handling leakage. We follow the "only computations leaks" (OCL) paradigm of [42]. That is, we assume that an adversary can only obtain information via leakage about the elements used for the particular computation. Thus, to represent the leakage of P_j during the execution of the ith round, we define the leakage function L^j_i whose inputs are $(\mathsf{pp}, \mathsf{st}^j_{i-1}, x^j, R^j_i, \mathsf{msg}_{i-1})$; a similar function L^j_{out} is used to capture the leakage of the output computation. Switching to the sub-computation level, the leakage obtained during the execution of the $i.i'$th computation is captured by the function $\mathsf{L}_{i.i'}$, whose inputs are $(\mathsf{pp}, \mathsf{st}^j_{i.i'-1}, x^j, R^j_{i.i'}, \mathsf{msg}_{i-1})$. So the total leakage of the ith round played by P_j is defined by $\mathsf{L}^j_i := (\{\mathsf{L}^j_{i.i'}\}_{i' \in [n_i]}\}$. The leakage resulting from the entire calculations performed by P_j is given by the set of leakage functions

$$\mathsf{L}^j := (\mathsf{L}^j_1, \dots, \mathsf{L}^j_{n_\pi}, \mathsf{L}^j_{out}) = (\{\mathsf{L}^j_{1.i'}\}_{i' \in [n_1]}, \dots, \{\mathsf{L}^j_{n_\pi.i'}\}_{i' \in [n_{n_\pi}]}, \mathsf{L}^j_{out}).$$

Recall that the inputs of the leakage function L^j_i are $(x^j, \mathsf{msg}_{i-1}, \mathsf{pp}, \mathsf{st}^j_{i.i'-1}, R^j_{i.i'})$. But, according to the "Only Computation Leaks" (OCL) model [42], *only the contents of the input cells that are actually accessed during the computation are leaked.* So, let $x^j_{|i}$ be the part of x^j used during the ith round, and $x^j_{|i.i'}$ be the part that is used during the $i.i'$ sub-computation. We can then use x^j instead of $x^j_{|i}$ in L^j_i, and instead of $x^j_{|i.i'}$ in $\mathsf{L}^j_{i.i'}$. (For example, the sender in an OT protocol may not use its input, (m_0, m_1) in every round. It can also use only m_0 or m_1 in a sub-computation.)

States are handled similarly to the modeling of the inputs. Specifically, let $\mathsf{st}^j_{i-1|i}$ be the values of the state memory cells accessed for the computation of the ith round, and $\mathsf{st}^j_{i.i'-1|i.i'}$ of the $i.i'$ computation. We replace st^j_{i-1} with $\mathsf{st}^j_{i-1|i}$ in L^j_i and $\mathsf{st}^j_{i.i'-1}$ with $\mathsf{st}^j_{i.i'-1|i.i'}$ in $\mathsf{L}^j_{i.i'}$, respectively. Finally, we note that our state model gives the same result as a model when the values carried by memory cells can be deleted: owing to their deletion, these values cannot be leaked [4]. As stated above, our motivation is a cleaner syntax.

[3] Should not be confuse with secret-sharing, here a share merely describe part of the message

[4] In our model, these values are not deleted, but since the corresponding memory cells are not accessed anymore, they cannot be leaked.

3.2 Security Definitions

In this section, we define security for a protocol that executes N instances of OTs in the presence of a side-channel adversary. Our goal is to protect the privacy of the sender. We consider two classes of attacks: (1) *intercepting side-channel adversaries*, and (2) *corrupted receiver side-channel adversaries*, as defined below.

Definition 13. *An* intercepting side-channel adversary (OT–S–LA) against the sender of an N-OT protocol *has access to the protocol transcript and the leakage of the sender's execution.*

Definition 14. *A* corrupted receiver side-channel adversary (OT–S–LR) against the sender of an N-OT protocol*, semi-honestly corrupts the receiver at the start of the execution and has access to the leakage of the sender's execution.*

We provide the security definitions for these two attacks. We assume that the inputs of all these N instances are chosen at the same time, after the setup phase of the protocol.

Let M be the input of S and r be the input of R. To simplify the notation we use S_i, $\mathsf{S}_{i.i'}$, R_i, $\mathsf{R}_{i.i'}$, to denote π_i^{S}, $\pi_{i.i'}^{\mathsf{S}}$, π_i^{R}, $\pi_{i.i'}^{\mathsf{R}}$, respectively. Without loss of generality, we assume that the sender speaks first.

Sender privacy against intercepting adversaries. We provide an indistinguishability definition, where the adversary cannot distinguish between two pairs of N inputs of S, even in the presence of leakage from the sender's computation and the protocol's transcript. Our security definition is inspired by CPAL (Def. 8) definition because it captures the indistinguishability of encrypting two different messages. We define security in the presence of a strong adversary that knows all the inputs except for one bit of information: it knows the receiver's input and two choices of the sender's inputs, but it does not know which input is used. Moreover, as for CPAL, the adversary gets the sender's leakage. Formally

Definition 15. *A* 1*-out-of-*2-OT *protocol* π *with* N *instances and* n_π *rounds whose execution has been divided into the* $\{\pi_{i.i'}\}_{i.i'\in\mathcal{I}}$ *computations, is said to be* (t,ϵ)-leakage-resistant for the sender in the presence of an intercepting side-channel adversary *(*OT–S–LA*) in the presence of leakage* $\mathsf{L} = (\mathsf{L}_{set}, \{\mathsf{L}_{i.i'}^{\mathsf{S}}\}_{i.i'\in\mathcal{I}}, \mathsf{L}_{out}^{\mathsf{S}})$*, if for any* t*-adversary* $\mathsf{A}^{\mathsf{L}} = (\mathsf{A}_1^{\mathsf{L}}, \mathsf{A}_2^{\mathsf{L}})$*,* $|\Pr[N\text{–}\mathsf{OT}\text{–}\mathsf{S}\text{–}\mathsf{LA}^{\mathsf{CPA}}_{\mathsf{OT}^{\mathsf{L}},\mathsf{A}}(1^\kappa) = 1] - 1/2| \leq \epsilon$*, with the* $N\text{–}\mathsf{OT}\text{–}\mathsf{S}\text{–}\mathsf{LA}_{\mathsf{OT}^{\mathsf{L}},\mathsf{A}}(1^\kappa)$*-experiment defined in Tab. 1.*

Security against corrupt receiver. Now, we consider a stronger adversary that also (semi-honestly) corrupts the receiver. This gives it access to all the internal states, randomness, and output of the receiver. Consequently, it can only choose the sender's inputs that match the output of the corrupted receiver, i.e. $m_{i,r_i}^0 = m_{i,r_i}^1\ \forall i$. Formally:

$\mathsf{pp} \leftarrow \pi.\mathsf{Setup}(1^\kappa, R^{\mathsf{S}}_{set}, R^{\mathsf{R}}_{set})$	$\mathsf{OTL}(\mathsf{pp}, M^b, r, \mathsf{L})$:
$\ell_{set} := \mathsf{L}_{set}(1^\kappa, R^{\mathsf{S}}_{set}, R^{\mathsf{R}}_{set})$	$M = M^b$
$(\mathsf{st}, M^0, M^1, r) \leftarrow \mathsf{A}^{\mathsf{L}}_1(\mathsf{pp}, \ell_{set})$ with	$i = 1$
$\quad r \in \{0,1\}^N$,	While $i \leq n_\pi$
$\quad M^0 = m^0_{1,0}, m^0_{1,1}, \ldots, m^0_{N,0}, m^0_{N,1}$	$\quad$ For $i' \in [n_i]$ //Sender's Round
$\quad M^1 = m^1_{1,0}, m^1_{1,1}, \ldots, m^1_{N,0}, m^1_{N,1}$,	$\quad\quad (\mathsf{pmsg}_{i.i'}, \mathsf{st}^{\mathsf{S}}_{i.i'}) \leftarrow \mathsf{S}_{i.i'}(\mathsf{pp}, \mathsf{st}^{\mathsf{S}}_{i.i'-1}, M, R^{\mathsf{S}}_{i.i'}, \mathsf{msg}_{i-1})$
$\quad \lvert m^0_{i,0}\rvert = \lvert m^0_{i,1}\rvert \; \forall i \in [N]$,	$\quad\quad \ell^{\mathsf{S}}_{i.i'} \leftarrow \mathsf{L}^{\mathsf{S}}_{i.i'}(\mathsf{pp}, \mathsf{st}^{\mathsf{S}}_{i.i'-1\mid i.i'}, M_{\mid i.i'}, R^{\mathsf{S}}_{i.i'}, \mathsf{msg}_{i-1})$
$\quad \lvert m^0_{i,j}\rvert = \lvert m^1_{i,j}\rvert \; \forall i, j \in [N] \times \{0,1\}$	$\quad \mathsf{msg}_i = \mathsf{pmsg}_{i.1} \Vert \ldots \Vert \mathsf{pmsg}_{i.n_i}$
$b \overset{\$}{\leftarrow} \{0,1\}$	$\quad i = i + 1$
$(\mathsf{trans}, \ell^{\mathsf{S}}) \leftarrow \mathsf{OTL}(\mathsf{pp}, M^b, r, \mathsf{L})$	$\quad$ For $i' \in [n_{i+1}]$ //Receiver's Round
$b' \leftarrow \mathsf{A}^{\mathsf{L}}_2(\mathsf{pp}, \mathsf{st}, \mathsf{trans}, \ell^{\mathsf{S}})$	$\quad\quad (\mathsf{pmsg}_{i.i'}, \mathsf{st}^{\mathsf{R}}_{i.i'}) \leftarrow \mathsf{R}_{i.i'}(\mathsf{pp}, \{\mathsf{st}^{\mathsf{R}}_{i.i'-1}, r, R^{\mathsf{R}}_{i.i'}, \mathsf{msg}_{i-1})$
If $b = b'$	$\quad \mathsf{msg}_i = \mathsf{pmsg}_{i.1} \Vert \ldots \Vert \mathsf{pmsg}_{i.n_i}$
$\quad$ Return 1	$\quad i = i + 1$
Return 0	$\ell^{\mathsf{S}}_{out} = \mathsf{L}^{\mathsf{S}}_{out}(\mathsf{pp}, \mathsf{st}^{\mathsf{S}}_{n_\pi.n_{n_\pi}}, R^{\mathsf{S}}_{out}, \mathsf{msg}_{n_\pi})$
	$\mathsf{trans} = (\mathsf{msg}_1, \ldots, \mathsf{msg}_{n_\pi}),\ \ell^{\mathsf{S}} = (\{\ell^{\mathsf{S}}_{i.i'}\}_{i \in \mathcal{I}}, \ell^{\mathsf{S}}_{out})$
	Return $(\mathsf{trans}, \ell^{\mathsf{S}})$

Table 1. The N-OT–S–LA experiment.

Definition 16. *A* 1*-out-of-*2-OT *protocol* π *with* N *instances and* n_π *rounds whose execution has been divided into the* $\{\pi_{i.i'}\}_{i.i' \in \mathcal{I}}$ *computations, is said to be* (t, ϵ)-leakage-resistant for the sender in the presence of a corrupt receiver side-channel adversary *(*OT-S-LR*) in the presence of leakage* $\mathsf{L} = (\mathsf{L}_{set}, \{\mathsf{L}^{\mathsf{S}}_{i.i'}\}_{i.i' \in \mathcal{I}}, \mathsf{L}^{\mathsf{S}}_{out})$*, if for any* t*-adversary* $\mathsf{A}^{\mathsf{L}} = (\mathsf{A}^{\mathsf{L}}_1, \mathsf{A}^{\mathsf{L}}_2)$,

$$|\Pr[N\text{-}\mathsf{OT}\text{-}\mathsf{S}\text{-}\mathsf{LR}^{\mathsf{CPA}}_{\mathsf{OT}^{\mathsf{L}},\mathsf{A}}(1^\kappa) = 1] - 1/2| \leq \epsilon,$$

with the N-OT-S-$\mathsf{LR}_{\mathsf{OT}^{\mathsf{L}},\mathsf{A}}(1^\kappa)$*-experiment defined in Tab. 2.*

4 A Leakage-Resilient Single OT Protocol

In this section, we will describe a single OT protocol and its security in the presence of leakage. We have already given an overview in Sec. 1.2, and here we provide its simulation-based (black-box) security without leakage in Sec. 4.1, and its security in the presence of leakage in Sec. 4.2.

4.1 Black-Box Simulation-Based Security

In this section, we introduce our protocol and then prove its simulation-based security (Def. 11) in the standard setting (that is, black-box, i.e., without leakage). We first list its building blocks and their (black-box) security properties:

i a key-exchange protocol π^{KE} (Def. 12 and Def. 10),
ii a PRF F (Def. 4),
iii an Eav-secure encryption scheme ENC (Def. 7 and Def. 8),
iv an oblivious transfer protocol π^{OT}(Sec. 2.6 and Def. 11).

```
$\mathsf{pp} \leftarrow \pi.\mathsf{Setup}(1^\kappa, R^{\mathsf{S}}_{set}, R^{\mathsf{R}}_{set})$
$\ell_{set} := \mathsf{L}_{set}(1^\kappa, R^{\mathsf{S}}_{set}, R^{\mathsf{R}}_{set})$
$(\mathsf{st}, M^0, M^1, r) \leftarrow \mathsf{A}^{\mathsf{L}}_1(\mathsf{pp}, (\ell_{set}, , R^{\mathsf{R}}_{set}))$
    with $r \in \{0,1\}^N$,
    $M^0 = m^0_{1,0}, m^0_{1,1}, \ldots, m^0_{N,0}, m^0_{N,1}$
    $M^1 = m^1_{1,0}, m^1_{1,1}, \ldots, m^1_{N,0}, m^1_{N,1}$,
    $|m^0_{i,0}| = |m^0_{i,1}|$,  $m^0_{i,r_i} = m^1_{i,r_i}$ $\forall i \in [N]$,
    $|m^0_{i,j}| = |m^1_{i,j}|$ $\forall i,j \in [N] \times \{0,1\}$
$b \xleftarrow{\$} \{0,1\}$
$(\mathsf{trans}, \ell^{\mathsf{S}}, M^{\mathsf{R}}, R^{\mathsf{R}}, \mathsf{st}^{\mathsf{R}}_{fin}) \leftarrow$
    $\mathsf{OTL\text{-}Rc}(\mathsf{pp}, M^b, r, \mathsf{L})$
$b' \leftarrow \mathsf{A}^{\mathsf{L}}_2(\mathsf{pp}, \mathsf{st}, (\mathsf{trans}, \ell^{\mathsf{S}}, M^{\mathsf{R}}, R^{\mathsf{R}}, \mathsf{st}^{\mathsf{R}}_{fin}))$
If $b = b'$
    Return 1
Return 0
```

```
$\mathsf{OTL\text{-}Rc}(\mathsf{pp}, M^b, r, \mathsf{L})$:
$M = M^b$
$i = 1$
While $i \leq n_\pi$
    For $i' \in [n_i]$                                        //Sender's Round
        $(\mathsf{pmsg}_{i.i'}, \mathsf{st}^{\mathsf{S}}_{i.i'}) \leftarrow \mathsf{S}_{i.i'}(\mathsf{pp}, \mathsf{st}^{\mathsf{S}}_{i.i'-1}, M, R^{\mathsf{S}}_{i.i'}, \mathsf{msg}_{i-1})$
        $\ell^{\mathsf{S}}_{i.i'} \leftarrow \mathsf{L}^{\mathsf{S}}_{i.i'}(\mathsf{pp}, \mathsf{st}^{\mathsf{S}}_{i.i'-1|i.i'}, M_{|i.i'}, R^{\mathsf{S}}_{i.i'}, \mathsf{msg}_{i-1})$
    $\mathsf{msg}_i = \mathsf{pmsg}_{i.1} \| \ldots \| \mathsf{pmsg}_{i.n_i}$
    $i = i + 1$
    For $i' \in [n_{i+1}]$                                    //Receiver's Round
        $(\mathsf{pmsg}_{i.i'}, \mathsf{st}^{\mathsf{R}}_{i.i'}) \leftarrow \mathsf{R}_{i.i'}(\mathsf{pp}, \{\mathsf{st}^{\mathsf{R}}_{i.i'-1}, r, R^{\mathsf{R}}_{i.i'}, \mathsf{msg}_{i-1})$
    $\mathsf{msg}_i = \mathsf{pmsg}_{i.1} \| \ldots \| \mathsf{pmsg}_{i.n_i}$
    $i = i + 1$
$M^{\mathsf{R}} = \pi^{\mathsf{R}}_{out}(\mathsf{pp}, \mathsf{st}^{\mathsf{R}}_{n_\pi.n_{n_\pi}}, R^{\mathsf{R}}_{out}, \mathsf{msg}_{n_\pi})$, $\mathsf{st}^{\mathsf{R}}_{fin} := \mathsf{st}^{\mathsf{R}}_{n_\pi.n_{n_\pi}}$
$\ell^{\mathsf{S}}_{out} = \mathsf{L}^{\mathsf{S}}_{out}(\mathsf{pp}, \mathsf{st}^{\mathsf{S}}_{n_\pi.n_{n_\pi}}, R^{\mathsf{S}}_{out}, \mathsf{msg}_{n_\pi})$
$\mathsf{trans} = (\mathsf{msg}_1, \ldots, \mathsf{msg}_{n_\pi})$, $\ell = (\{\ell^{\mathsf{S}}_{i.i'}\}_{i \in \mathcal{I}}, \ell^{\mathsf{S}}_{out})$
$R^{\mathsf{R}} = (\{R^{\mathsf{R}}_{i.i'}\}_{i \in [n_\pi], i' \in [n_i]}, R^{\mathsf{R}}_{out})$
Return $(\mathsf{trans}, \ell^{\mathsf{S}}, M^{\mathsf{R}}, R^{\mathsf{R}}, \mathsf{st}^{\mathsf{R}}_{fin})$
```

Table 2. The N-OT-S-LR experiment. OTL-Rc stands for OTL with the receiver corrupted. Changes from Tab. 1 are highlighted.

$\pi^{\mathsf{LR\text{-}OT}}$ proceeds as follows: 1) First, the sender and the receiver execute π^{KE} to both get k. Next 2a), S picks two random keys k_0, k_1, 2b) encrypts them with k to get y_0, y_1, where $y_i = \mathsf{F}_k(P_i) \oplus k_i$ and P_i is a public value ($P_i \neq P_j$). Then 2c), S refreshes its keys by generating $k_{i,E}$'s, where $k_{i,E} = \mathsf{F}_{k_i}(P_2)$ and P_2 is a public constant, and 2d) uses these keys to encrypt m_i, i.e., $c_i = \mathsf{Enc}_{k_{i,E}}(m_i)$. S 2e) sends these ciphertexts to R in the clear. Finally 3), S and R execute π^{OT} where S inputs (y_0, y_1) and R inputs r. The complete details can be found in Alg. 1 . The protocol's correctness follows from the correctness of its underlying building blocks.

Theorem 1. *Let π^{KE} be a $(t_1, \epsilon_{\mathsf{KE}})$-secure key-exchange protocol (Def. 10), let* F *be a $(1, t_2, \epsilon_{\mathsf{PRF}})$*-PRF *(Def. 4), let* ENC *be a $(t_3, \epsilon_{\mathsf{Eav}})$-eavesdropper-secure encryption scheme (Def. 8), and let π^{OT} be a protocol that $(t_4, \epsilon_{\mathsf{OT}})$-securely computes the* OT *functionality (Def. 11,Sec. 2.6), then, the $\pi^{\mathsf{LR\text{-}OT}}$ protocol defined in Alg. 1, (t, ϵ)-securely computes the* OT *functionality with*

$$\epsilon \leq \epsilon_{\mathsf{KE}} + \epsilon_{\mathsf{OT}} + \epsilon_{\mathsf{PRF}} + \epsilon_{\mathsf{Eav}},$$

provided that $|m_0|, |m_1| \leq B$ bits, where $t_1 = t + t_{\mathsf{OT}} + 4t_{\mathsf{F}} + 2t_{\mathsf{Enc}}$, $t_2 = t + t_{\mathsf{Sim,KE}} + t_{\mathsf{Sim,OT}} + 4t_{\mathsf{F}} + 2t_{\mathsf{Enc}}$, $t_3 = t + t_{\mathsf{Sim,KE}} + t_{\mathsf{Sim,OT}} + 2t_{\mathsf{F}} + t_{\mathsf{Enc}}$, $t_4 = t + t_{\mathsf{Sim,KE}} + t_{\mathsf{Sim,OT}} + 2t_{\mathsf{F}} + t_{\mathsf{Enc}}$, $t_{\mathsf{Sim,KE}}$ is the time needed to simulate the π^{KE} protocol, t_{OT} to execute the π^{OT} protocol, $t_{\mathsf{Sim,OT}}$ to simulate the π^{OT} protocol, t_{F} to execute F*, and t_{Enc} to encrypt with* Enc *a message of at most B bits.*

Intuitively, if the sender is (semi-honestly) corrupted, it will only see messages exchanged computed by the π^{KE} and π^{OT} protocols. Using the simulators for these two protocols, we simulate the sender's view. A (semi-honest) corrupted receiver receives, in addition to the messages exchanged by the π^{KE} and π^{OT} protocols, the ciphertexts c_0, c_1. Again, we use the simulators for these two protocols. To simulate π^{OT}, the simulator can honestly compute y_r picking a random k_r and computing $y_r = \mathsf{F}_k(P_r) \oplus k_r$. Then, it computes $k_{r,E}$ and, thus, c_r, correctly. The simulator replaces the ciphertext $c_{\bar{r}}$ (the encryption of $m_{\bar{r}}$, i.e., the message R does not receive) with the encryption of an arbitrary message (since ENC is Eav-secure and $k_{\bar{r},E}$ is a random key because F is a PRF and $y_{\bar{r}}$ is never seen by the receiver). Now, we give the complete statement (Thm. 1) with the quantitative bounds and the proof.

Proof. We start by building the simulator for the sender, $\mathsf{Sim}_{\mathsf{S}}$ and by proving that the view given by $\mathsf{Sim}_{\mathsf{S}}$ is (t, ϵ)-computationally indistinguishable from the real view. Then, we build $\mathsf{Sim}_{\mathsf{R}}$, and we prove that it is a (t, ϵ)-simulator.

Notations. Let $M = (m_0, m_1)$ be the input of the sender and r be the input of the receiver.

Since π^{KE} is $(t_1, \epsilon_{\mathsf{KE}})$-secure, there are two simulators $\mathsf{Sim}_{\mathsf{S}}^{\mathsf{KE}}$, $\mathsf{Sim}_{\mathsf{R}}^{\mathsf{KE}}$ s.t. the joint distribution of the $(\mathsf{view}_{\mathsf{S}}^{\pi^{\mathsf{KE}}}(1^\kappa, n_{\mathsf{ENC}}), k')$ [resp. $(\mathsf{view}_{\mathsf{R}}^{\pi^{\mathsf{KE}}}(1^\kappa, n_{\mathsf{ENC}}), k')$], where k' is the output of the π^{KE}-protocol, is ϵ_{KE}-computationally indistinguishable from $(\mathsf{Sim}_{\mathsf{S}}^{\mathsf{KE}}(1^\kappa, n_{\mathsf{ENC}}, k), k)$ [resp. $(\mathsf{Sim}_{\mathsf{R}}^{\mathsf{KE}}(1^\kappa, n_{\mathsf{Enc}}, k), k)$] with $k \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$. Similarly, since π^{OT} is $(t_4, \epsilon_{\mathsf{OT}})$-secure, there are two simulators $\mathsf{Sim}_{\mathsf{S}}^{\mathsf{OT}}$, $\mathsf{Sim}_{\mathsf{R}}^{\mathsf{OT}}$ s.t. $\mathsf{view}_{\mathsf{S}}^{\pi^{\mathsf{OT}}}(1^\kappa, (m_0, m_1), \perp)$ [resp. $(\mathsf{view}_{\mathsf{R}}^{\pi^{\mathsf{OT}}}(1^\kappa, r, m_r)]$ is ϵ_{OT}-computationally indistinguishable from $(\mathsf{Sim}_{\mathsf{S}}^{\mathsf{OT}}(1^\kappa, (x_0, x_1), \perp)$ [resp. $\mathsf{Sim}_{\mathsf{R}}^{\mathsf{OT}}(1^\kappa, r, x_r)$].

Sender (semi-honestly) corrupted.

We simulate the sender's view with $\mathsf{Sim}_{\mathsf{S}}(1^\kappa, (m_0, m_1), \perp)$. It works like this:

– **Setup-phase:** $\mathsf{Sim}_{\mathsf{S}}(1^\kappa)$ chooses n_{ENC}, F, ENC, and three different strings in $\{0,1\}^{n_{\mathsf{ENC}}}$: P_0, P_1, and P_2. Then it runs $\mathsf{Sim}_{\mathsf{S}}^{\mathsf{KE.Setup}}(1^\kappa, n_{\mathsf{ENC}})$ obtaining $(\mathsf{pp}^{\mathsf{KE}}, \mathsf{view}_{\mathsf{S}}^{\pi^{\mathsf{KE}}.\mathsf{Setup}})$, and $\mathsf{Sim}_{\mathsf{S}}^{\mathsf{OT.Setup}}(1^\kappa)$ obtaining $(\mathsf{pp}^{\mathsf{OT}}, \mathsf{view}_{\mathsf{S}}^{\pi^{\mathsf{OT}}.\mathsf{Setup}})$, respectively. $\mathsf{Sim}_{\mathsf{S}}$ defines
$$\mathsf{view}_{\mathsf{S}}^{\mathsf{Setup}} := ((\mathsf{pp}^{\mathsf{KE}}, \mathsf{pp}^{\mathsf{OT}}, n_{\mathsf{ENC}}, \mathsf{F}), \mathsf{view}_{\mathsf{S}}^{\pi^{\mathsf{KE}}.\mathsf{Setup}}, \mathsf{view}_{\mathsf{S}}^{\pi^{\mathsf{OT}}.\mathsf{Setup}}).$$
– **Key-exchange phase:** $\mathsf{Sim}_{\mathsf{S}}$ picks $k \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$ and runs $\mathsf{Sim}_{\mathsf{S}}^{\mathsf{KE}}(1^\kappa, \mathsf{pp}^{\mathsf{KE}}, k)$ obtaining $\mathsf{view}_{\mathsf{S}}^{\mathsf{KE.KE}}$.
– **Sender's phase:** Using S'random tape, $\mathsf{Sim}_{\mathsf{S}}$ can pick correctly $k_0, k_1 \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$. Then, it computes (y_0, y_1) as in Alg 1, $y_i = \mathsf{F}_k(P_i) \oplus k_i$.
– **Send key-phase:** $\mathsf{Sim}_{\mathsf{S}}$ computes $\mathsf{view}_{\mathsf{S}}^{\mathsf{OT}} := \mathsf{Sim}_{\mathsf{S}}^{\mathsf{OT.OT}}(1^\kappa, \mathsf{pp}^{\mathsf{OT}}, (y_0, y_1), \perp)$.
– **Simulator output:** At the end, $\mathsf{Sim}_{\mathsf{S}}$ outputs $\mathsf{view}_{\mathsf{S}}^{\mathsf{Sim}}$ where
$$\mathsf{view}_{\mathsf{S}}^{\mathsf{Sim}} := (\mathsf{view}_{\mathsf{S}}^{\mathsf{Setup}}, \mathsf{view}_{\mathsf{S}}^{\mathsf{KE.KE}}, \mathsf{view}_{\mathsf{S}}^{\mathsf{OT}}).$$

Hybrids. To prove that the simulator $\mathsf{Sim}_{\mathsf{S}}$ outputs a view that is indistinguishable from the real execution of the protocol, we introduce three different view distributions.

– Let $H_{\mathsf{S}}^0(1^\kappa, M, r)$ be S's view distribution of the real execution of $\pi^{\mathsf{LR\text{-}OT}}$.

– We modify the previous view distribution, obtaining the view distribution $H^1_{\mathsf{S}}(1^\kappa, M, r)$ by simulating the execution of the key-exchange protocol, using $\mathsf{Sim}^{\mathsf{KE}}_{\mathsf{S}}$, instead of having the real execution. We use $\mathsf{Sim}^{\mathsf{KE}}_{\mathsf{S}}(1^\kappa, n_{\mathsf{ENC}}, \mathsf{pp}^{\mathsf{KE}}, k)$, where 1^κ is the security parameter, , n_{ENC} obtained as in the real execution of the protocol and $k \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$ (π^{KE} is a key-exchange protocol (Def. 12), that is, it implements the functionality $k \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$).
– We modify the previous view distribution, obtaining the view distribution $H^2_{\mathsf{S}}(1^\kappa, M, r)$ by simulating the execution of the OT protocol, using $\mathsf{Sim}^{\mathsf{OT}}_{\mathsf{S}}(1^\kappa, \mathsf{pp}^{\mathsf{OT}}, (y_0, y_1), \perp)$, instead of having the real execution. $\mathsf{Sim}_{\mathsf{S}}$ can compute y_0 and y_1 correctly: accessing the random tape of S, $\mathsf{Sim}_{\mathsf{S}}$ can pick $k_0, k_1 \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$ correctly and compute $y_i = \mathsf{F}_k(P_i) \oplus k_i$.

Indistinguishability of the hybrids. Now, we show that each view distribution is computationally indistinguishable from the next.

Since π^{KE} is a $(t_1, \epsilon_{\mathsf{KE}})$-secure key-exchange protocol, then

$$|\Pr[1 \leftarrow \mathsf{A}(H^0_{\mathsf{S}}(1^\kappa, M, r))] - \Pr[1 \leftarrow \mathsf{A}(H^1_{\mathsf{S}}(1^\kappa, M, r))]| \leq \epsilon_{\mathsf{KE}}.$$

Since π^{OT} is a $(t_4, \epsilon_{\mathsf{OT}})$-secure OT protocol, and y_0 and y_1 are computed as in the correct execution of $\pi^{\mathsf{LR\text{-}OT}}$, then

$$|\Pr[1 \leftarrow \mathsf{A}(H^1_{\mathsf{S}}(1^\kappa, M, r))] - \Pr[1 \leftarrow \mathsf{A}(H^2_{\mathsf{S}}(1^\kappa, M, r))]| \leq \epsilon_{\mathsf{OT}}.$$

This concludes the proof since $H^2_{\mathsf{S}}(1^\kappa, M, r)$ is the distribution of the views simulated by $\mathsf{Sim}_{\mathsf{S}}$. Thus, the real view distribution, $H^0_{\mathsf{S}}(1^\kappa, M, r)$, and the simulated one, $H^2_{\mathsf{S}}(1^\kappa, M, r)$ are (t, ϵ')-computationally indistinguishable, since

$$|\Pr[1 \leftarrow \mathsf{A}(H^0_{\mathsf{S}}(1^\kappa, M, r))] - \Pr[1 \leftarrow \mathsf{A}(H^2_{\mathsf{S}}(1^\kappa, M, r))]| \leq \epsilon_{\mathsf{KE}} + \epsilon_{\mathsf{OT}} = \epsilon' \leq \epsilon.$$

Adversaries used in the hybrids' transitions. To finish the proof for the sender's corrupted case, we have to explicitly describe the adversaries used in the reduction between the hybrids:

– **The t'_1-adversary B against the π^{KE}:** B has to distinguish whether it interacts with an oracle providing a real execution of π^{KE} or a simulated one.
At the start of the game, B receives 1^κ from A. It chooses $n_{\mathsf{ENC}}, \mathsf{F}, \mathsf{ENC}, P_0, P_1, P_2$ as in the setup phase 1) and π^{OT}. Then, it runs its oracle on input 1^κ receiving the public parameters $\mathsf{pp}_{\mathsf{KE}}$, the output k_0 and the view $\mathsf{view}^{\mathsf{KE}}_{\mathsf{S}}$. B splits it into $\mathsf{view}^{\mathsf{KE.Setup}}_{\mathsf{S}}$ and $\mathsf{view}^{\mathsf{KE.KE}}_{\mathsf{S}}$. After that, B run the $\mathsf{OT.Setup}$ sub-protocol on input $(1^\kappa, n_{\mathsf{ENC}})$, obtaining $\mathsf{pp}^{\mathsf{OT}}$ and the $\mathsf{view}^{\pi^{\mathsf{OT.Setup}}}_{\mathsf{S}}$, and it sets

$$\mathsf{view}^{\mathsf{Setup}}_{\mathsf{S}} := ((\mathsf{pp}^{\mathsf{KE}}, \mathsf{pp}^{\mathsf{OT}}, n_{\mathsf{ENC}}, \mathsf{F}), \mathsf{view}^{\mathsf{KE.Setup}}_{\mathsf{S}}, \mathsf{view}^{\pi^{\mathsf{OT}}.\mathsf{Setup}}_{\mathsf{S}}),$$

which it sends to A. When A outputs (M, r), B picks $k_0, k_1 \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$, computes $z_0 = \mathsf{F}_k(P_0)$, $z_1 = \mathsf{F}_k(P_1)$, $y_0 = z_0 \oplus k_0$, $y_1 = z_1 \oplus k_1$. Finally, B executes the $\mathsf{OT.OT}$ sub-protocol where the public parameters are $\mathsf{pp}_{\mathsf{OT}}$, the sender has input (y_0, y_1), and the receiver has input r, obtaining the view $\mathsf{view}^{\mathsf{OT}}_{\mathsf{S}}$. In the end, since there is no output, B answers A the view

$$\mathsf{view}_{\mathsf{S}} = (\mathsf{view}^{\mathsf{KE}}_{\mathsf{S}}, \mathsf{view}^{\mathsf{OT}}_{\mathsf{S}}).$$

When A outputs its bit b, B sees it, and B outputs as its output bit. B runs in time $t + t_{\mathsf{OT}} + 2t_{\mathsf{F}} = t_1' \le t_1$.

If the oracle B interacts with, outputs a real execution of π^{OT}, B returns the H^0_{S}-view to A, otherwise the H^1_{S}-view.

– **The t_4'-adversary C against the π^{OT}:**
C has to distinguish whether it interacts with an oracle providing a real execution of π^{OT} or a simulated one.
At the start of the game, C receives 1^κ from A. It chooses $n_{\mathsf{ENC}}, \mathsf{F}, \mathsf{ENC}, P_0, P_1, P_2$ as in the setup phase 1) and π^{OT}. Then, it runs $\mathsf{Sim}^{\mathsf{KE}}_{\mathsf{S}}(1^\kappa)$ obtaining the public parameters $\mathsf{pp}_{\mathsf{KE}}$, and the view $\mathsf{view}^{\mathsf{KE.Setup}}_{\mathsf{S}}$ and it calls its oracle on input 1^κ to obtain $\mathsf{pp}^{\mathsf{OT}}$ and the $\mathsf{view}^{\pi^{\mathsf{OT.Setup}}}_{\mathsf{S}}$. C sets

$$\mathsf{view}^{\mathsf{Setup}}_{\mathsf{S}} := ((\mathsf{pp}^{\mathsf{KE}}, \mathsf{pp}^{\mathsf{OT}}, n_{\mathsf{ENC}}, \mathsf{F}), \mathsf{view}^{\mathsf{KE.Setup}}_{\mathsf{S}}, \mathsf{view}^{\pi^{\mathsf{OT}}.\mathsf{Setup}}_{\mathsf{S}}),$$

which it sends to A. When A outputs (M, r), C picks $k_0 \overset{\$}{\leftarrow}$ and running $\mathsf{Sim}^{\mathsf{KE.KE}}_{\mathsf{S}}(pp_{\mathsf{KE}}, k_0)$ obtaining $\mathsf{view}^{\mathsf{KE}}_{\mathsf{S}}$. Then, C picks $k_0, k_1 \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$, computes $z_0 = \mathsf{F}_k(P_0)$, $z_1 = \mathsf{F}_k(P_1)$, $y_0 = z_0 \oplus k_0$, $y_1 = z_1 \oplus k_1$. Finally, C queries its oracle on input $(\mathsf{pp}_{\mathsf{OT}}, (y_0, y_1))$, obtaining the view $\mathsf{view}^{\mathsf{OT}}_{\mathsf{S}}$. At the end, C answers A the view

$$\mathsf{view}_{\mathsf{S}} = (\mathsf{view}^{\mathsf{KE}}_{\mathsf{S}}, \mathsf{view}^{\mathsf{OT}}_{\mathsf{S}}).$$

When A outputs its bit b, C sees it, and C outputs as its output bit. C runs in time $t + t_{\mathsf{Sim,KE}} + 2t_{\mathsf{F}} = t_4' \le t_4$.

If the oracle C interacts with, outputs a real execution of π^{OT}, C returns the H^1_{S}-view to A, otherwise the H^2_{S}-view.

Receiver (semi-honestly) corrupted.

We simulate the receiver's view with $\mathsf{Sim}_{\mathsf{R}}(1^\kappa, r, m_r)$. It works like this:

– **Setup-phase:** $\mathsf{Sim}_{\mathsf{R}}(1^\kappa)$ chooses n_{ENC}, F, ENC, and three different strings in $\{0,1\}^{n_{\mathsf{ENC}}}$: P_0, P_1, and P_2. Then it runs $\mathsf{Sim}^{\mathsf{KE.Setup}}_{\mathsf{R}}(1^\kappa, n_{\mathsf{ENC}})$ obtaining $(\mathsf{pp}^{\mathsf{KE}}, \mathsf{view}^{\pi^{\mathsf{KE}}.\mathsf{Setup}}_{\mathsf{R}})$, and $\mathsf{Sim}^{\mathsf{OT.Setup}}_{\mathsf{R}}(1^\kappa)$ obtaining $(\mathsf{pp}^{\mathsf{OT}}, \mathsf{view}^{\pi^{\mathsf{OT}}.\mathsf{Setup}}_{\mathsf{R}})$, respectively. $\mathsf{Sim}_{\mathsf{R}}$ defines

$$\mathsf{view}^{\mathsf{Setup}}_{\mathsf{R}} := ((\mathsf{pp}^{\mathsf{KE}}, \mathsf{pp}^{\mathsf{OT}}, n_{\mathsf{ENC}}, \mathsf{F}), \mathsf{view}^{\pi^{\mathsf{KE}}.\mathsf{Setup}}_{\mathsf{R}}, \mathsf{view}^{\pi^{\mathsf{OT}}.\mathsf{Setup}}_{\mathsf{R}}).$$

– **Key-exchange phase:** $\mathsf{Sim}_{\mathsf{R}}$ picks $k \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$ and runs $\mathsf{Sim}^{\mathsf{KE}}_{\mathsf{R}}(1^\kappa, \mathsf{pp}^{\mathsf{KE}}, k)$ obtaining $\mathsf{view}^{\mathsf{KE.KE}}_{\mathsf{R}}$.
– **Sender's phase:** $\mathsf{Sim}_{\mathsf{R}}$ picks $k_r, k_{\bar{r},E} \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$, and an arbitrary message $\tilde{m}_{\bar{r}}$ (for example, $\tilde{m}_{\bar{r}} \overset{\$}{\leftarrow} \{0,1\}^{|m_r|}$). It computes $y_r = \mathsf{F}_k(P_r) \oplus k_r$, $k_{r,E} = \mathsf{F}_{k_r}(P_2), c_r = \mathsf{Enc}_{k_{r,E}}(m_r)$, $c_{\bar{r}} = \mathsf{Enc}_{k_{\bar{r},E}}(\tilde{m}_{\bar{r}})$.
– **Send key-phase:** $\mathsf{Sim}_{\mathsf{R}}$ computes $\mathsf{view}^{\mathsf{OT}}_{\mathsf{R}} := \mathsf{Sim}^{\mathsf{OT.OT}}_{\mathsf{R}}(1^\kappa, \mathsf{pp}^{\mathsf{OT}}, r, y_r)$.
– **Simulator output:** At the end, $\mathsf{Sim}_{\mathsf{R}}$ outputs

$$\mathsf{view}^{\mathsf{Sim}}_{\mathsf{R}} := (\mathsf{view}^{\mathsf{Setup}}_{\mathsf{R}}, \mathsf{view}^{\mathsf{KE.KE}}_{\mathsf{R}}, c_0, c_1, \mathsf{view}^{\mathsf{OT}}_{\mathsf{R}}).$$

Hybrids. To prove that the simulator $\mathsf{Sim}_{\mathsf{R}}$ outputs a view that is indistinguishable from the real execution of the protocol, we introduce five different view distributions.

- Let $H^0_{\mathsf{R}}(1^\kappa, M, r)$ be R's view distribution of the real execution of $\pi^{\mathsf{LR\text{-}OT}}$.
- We modify the previous view distribution, obtaining the view distribution $H^1_{\mathsf{R}}(1^\kappa, M, r)$ by simulating the execution of the key-exchange protocol, using $\mathsf{Sim}^{\mathsf{KE}}_{\mathsf{R}}$, instead of having the real execution. We use $\mathsf{Sim}^{\mathsf{KE}}_{\mathsf{R}}(1^\kappa, n_{\mathsf{ENC}}, k)$, where 1^κ is the security parameter, , n_{ENC} obtained as in the real execution of the protocol and $k \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$ (π^{KE} is a key-exchange protocol (Def. 12), that is, it implements the functionality $k \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$).
- We modify the previous view distribution, obtaining the view distribution $H^2_{\mathsf{R}}(1^\kappa, M, r)$ by simulating the execution of the OT protocol, using $\mathsf{Sim}^{\mathsf{OT}}_{\mathsf{R}}$, instead of having the real execution. We use $\mathsf{Sim}^{\mathsf{OT}}_{\mathsf{R}}(1^\kappa, r, y_r)$, where 1^κ is the security parameter, $y_r = \mathsf{F}_k(P_r) \oplus k_r$, with $k_r \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$.
- We modify the previous view distribution obtaining the view distribution $H^3_{\mathsf{R}}(1^\kappa, M, r)$ by picking $k_{\bar{r},E} \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$, instead of computing $k_{\bar{r},E} = \mathsf{F}_{k_{\bar{r}}}(P_2)$.
- We modify the previous view distribution obtaining the view distribution $H^4_{\mathsf{R}}(1^\kappa, M, r)$ by computing $c_{\bar{r}} = \mathsf{Enc}_{k_{\bar{r},E}}(\tilde{m}_{\bar{r}})$, where $\tilde{m}_{\bar{r}}$ is an arbitrary message (for example, $\tilde{m}_{\bar{r}} \xleftarrow{\$} \{0,1\}^{|m_r|}$).

Indistinguishability of the hybrids. Now, we show that each view distribution is computationally indistinguishable from the next.

Since π^{KE} is a $(t_1, \epsilon_{\mathsf{KE}})$-secure key-exchange protocol, then

$$|\Pr[1 \leftarrow \mathsf{A}(H^0_{\mathsf{R}}(1^\kappa, M, r))] - \Pr[1 \leftarrow \mathsf{A}(H^1_{\mathsf{R}}(1^\kappa, M, r))]| \leq \epsilon_{\mathsf{KE}}.$$

Since π^{OT} is a $(t_4, \epsilon_{\mathsf{OT}})$-secure OT protocol, $\mathsf{Sim}_{\mathsf{R}}$ picks k_r with the same distribution as in the original protocol and computes y_r as in the correct execution of $\pi^{\mathsf{LR\text{-}OT}}$, then

$$|\Pr[1 \leftarrow \mathsf{A}(H^1_{\mathsf{R}}(1^\kappa, M, r))] - \Pr[1 \leftarrow \mathsf{A}(H^2_{\mathsf{R}}(1^\kappa, M, r))]| \leq \epsilon_{\mathsf{OT}}.$$

Since F is a $(1, t_2, \epsilon_{\mathsf{PRF}})$-$\mathsf{PRF}$ and $k_{\bar{r}}$ is picked randomly and it is secret for the receiver, then

$$|\Pr[1 \leftarrow \mathsf{A}(H^2_{\mathsf{R}}(1^\kappa, M, r))] - \Pr[1 \leftarrow \mathsf{A}(H^3_{\mathsf{R}}(1^\kappa, M, r))]| \leq \epsilon_{\mathsf{PRF}}.$$

Since ENC is a $(t_3, \epsilon_{\mathsf{Eav}})$-$\mathsf{Eav}$-secure encryption scheme and $k_{\bar{r},E}$ is picked randomly, and it is secret for the receiver, then

$$|\Pr[1 \leftarrow \mathsf{A}(H^3_{\mathsf{R}}(1^\kappa, M, r))] - \Pr[1 \leftarrow \mathsf{A}(H^4_{\mathsf{R}}(1^\kappa, M, r))]| \leq \epsilon_{\mathsf{Eav}}.$$

This concludes the proof since $H^4_{\mathsf{R}}(1^\kappa, M, r)$ is the distribution of the views simulated by $\mathsf{Sim}_{\mathsf{R}}$. Thus, the real view distribution, $H^0_{\mathsf{R}}(1^\kappa, M, r)$, and the simulated one, $H^4_{\mathsf{R}}(1^\kappa, M, r)$ are (t, ϵ)-computationally indistinguishable, since

$$|\Pr[1 \leftarrow \mathsf{A}(H^0_{\mathsf{R}}(1^\kappa, M, r))] - \Pr[1 \leftarrow \mathsf{A}(H^4_{\mathsf{R}}(1^\kappa, M, r))]| \leq \epsilon_{\mathsf{KE}} + \epsilon_{\mathsf{OT}} + \epsilon_{\mathsf{PRF}} + \epsilon_{\mathsf{Eav}} = \epsilon.$$

Adversaries used in the hybrids' transitions. To finish the proof for the receiver's corrupted case, we have to explicitly describe the adversaries used in the reduction between the hybrids:

- **The t_1-adversary B against the π^{KE}:** B has to distinguish whether it interacts with an oracle providing a real execution of π^{KE} or a simulated one.

  At the start of the game, B receives 1^κ from A. It chooses $n_{\mathsf{ENC}}, \mathsf{F}, \mathsf{ENC}, P_0, P_1, P_2$ as in the setup phase 1) and π^{OT}. Then, it runs its oracle on input 1^κ receiving the public parameters $\mathsf{pp}_{\mathsf{KE}}$, the output k_0 and the view $\mathsf{view}_{\mathsf{R}}^{\mathsf{KE}}$. B splits it into $\mathsf{view}_{\mathsf{R}}^{\mathsf{KE.Setup}}$ and $\mathsf{view}_{\mathsf{R}}^{\mathsf{KE.KE}}$. After that, B run the $\mathsf{OT.Setup}$ sub-protocol obtaining $\mathsf{pp}^{\mathsf{OT}}$ and the $\mathsf{view}_{\mathsf{R}}^{\pi^{\mathsf{OT.Setup}}}$ on input $(1^\kappa, n_{\mathsf{ENC}})$ and it sets

  $$\mathsf{view}_{\mathsf{R}}^{\mathsf{Setup}} := ((\mathsf{pp}^{\mathsf{KE}}, \mathsf{pp}^{\mathsf{OT}}, n_{\mathsf{ENC}}, \mathsf{F}), \mathsf{view}_{\mathsf{R}}^{\mathsf{KE.Setup}}, \mathsf{view}_{\mathsf{R}}^{\pi^{\mathsf{OT}}.\mathsf{Setup}}),$$

  which it sends to A. When A outputs (M, r), B picks $k_0, k_1 \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$, computes $z_0 = \mathsf{F}_k(P_0)$, $z_1 = \mathsf{F}_k(P_1)$, $y_0 = z_0 \oplus k_0$, $y_1 = z_1 \oplus k_1$. Moreover, B computes $k_{0,E} = \mathsf{F}_{k_0}(P_2)$, $k_{1,E} = \mathsf{F}_{k_1}(P_2)$, and the ciphertexts $c_0 = \mathsf{Enc}_{k_{0,E}}(m_0)$, and $c_1 = \mathsf{Enc}_{k_{1,E}}(m_1)$. Finally, B executes the $\mathsf{OT.OT}$ sub-protocol where the sender has input (y_0, y_1), and the receiver has input r, obtaining the view $\mathsf{view}_{\mathsf{R}}^{\mathsf{OT}}$. At the end, B answers A the view

  $$\mathsf{view}_{\mathsf{R}} = (\mathsf{view}_{\mathsf{R}}^{\mathsf{KE}}, (c_0, c_1), \mathsf{view}_{\mathsf{R}}^{\mathsf{OT}}).$$

  When A outputs its bit b, B sees it, and B outputs as its output bit. B runs in time $t + t_{\mathsf{OT}} + 4t_{\mathsf{F}} + 2t_{\mathsf{Enc}} \leq t_1$.

  If the oracle B interacts with, outputs a real execution of π^{KE}, B returns the H_{R}^0-view to A, otherwise the H_{R}^1-view.

- **The t_4-adversary C against the π^{OT}:** C has to distinguish whether it interacts with an oracle providing a real execution of π^{OT} or a simulated one.

  At the start of the game, C receives 1^κ from A. It chooses $n_{\mathsf{ENC}}, \mathsf{F}, \mathsf{ENC}, P_0, P_1, P_2$ as in the setup phase 1). Then, it runs $\mathsf{Sim}_{\mathsf{R}}^{\mathsf{KE}}(1^\kappa)$, receiving the public parameters $\mathsf{pp}_{\mathsf{KE}}$, the output k_0 and the view $\mathsf{view}_{\mathsf{R}}^{\mathsf{KE}}$. C splits it into $\mathsf{view}_{\mathsf{R}}^{\mathsf{KE.Setup}}$ and $\mathsf{view}_{\mathsf{R}}^{\mathsf{KE.KE}}$. After that, C calls its oracle on input $(1^\kappa, n_{\mathsf{ENC}})$, obtaining $\mathsf{pp}^{\mathsf{OT}}$ and the $\mathsf{view}_{\mathsf{R}}^{\pi^{\mathsf{OT.Setup}}}$, and it sets

  $$\mathsf{view}_{\mathsf{R}}^{\mathsf{Setup}} := ((\mathsf{pp}^{\mathsf{KE}}, \mathsf{pp}^{\mathsf{OT}}, n_{\mathsf{ENC}}, \mathsf{F}), \mathsf{view}_{\mathsf{R}}^{\mathsf{KE.Setup}}, \mathsf{view}_{\mathsf{R}}^{\pi^{\mathsf{OT}}.\mathsf{Setup}}),$$

  which it sends to A. When A outputs (M, r), C picks $k_0, k_1 \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$, computes $z_0 = \mathsf{F}_k(P_0)$, $z_1 = \mathsf{F}_k(P_1)$, $y_0 = z_0 \oplus k_0$, $y_1 = z_1 \oplus k_1$. Moreover, C computes $k_{0,E} = \mathsf{F}_{k_0}(P_2)$, $k_{1,E} = \mathsf{F}_{k_1}(P_2)$, and the ciphertexts $c_0 = \mathsf{Enc}_{k_{0,E}}(m_0)$, and $c_1 = \mathsf{Enc}_{k_{1,E}}(m_1)$. Finally, C calls its oracle on input $(\mathsf{pp}_{\mathsf{OT}}, r, y_r)$, obtaining the view $\mathsf{view}_{\mathsf{R}}^{\mathsf{OT}}$. At the end, C answers A the view

  $$\mathsf{view}_{\mathsf{R}} = (\mathsf{view}_{\mathsf{R}}^{\mathsf{KE}}, (c_0, c_1), \mathsf{view}_{\mathsf{R}}^{\mathsf{OT}}).$$

  When A outputs its bit b, C sees it, and C outputs as its output bit. C runs in time $t + t_{\mathsf{Sim,KE}} + 4t_{\mathsf{F}} + 2t_{\mathsf{Enc}} \leq t_4$.

  If the oracle C interacts with, outputs a real execution of π^{OT}, C returns the H_{R}^1-view to A, otherwise the H_{R}^2-view.

– **The $(1, t_2)$-adversary D against the PRF F:** D has to distinguish whether it interacts with an oracle outputting random values or with one using $\mathsf{F}_{k_{\bar{r}}}$. At the start of the game, D receives 1^κ from A. It chooses $n_{\mathsf{ENC}}, \mathsf{ENC}, P_0, P_1, P_2$ as in the setup phase 1). Then, it runs $\mathsf{Sim}_{\mathsf{R}}^{\mathsf{KE}}(1^\kappa)$, receiving the public parameters $\mathsf{pp}_{\mathsf{KE}}$, the output k_0 and the view $\mathsf{view}_{\mathsf{R}}^{\mathsf{KE}}$. D splits it into $\mathsf{view}_{\mathsf{R}}^{\mathsf{KE.Setup}}$ and $\mathsf{view}_{\mathsf{R}}^{\mathsf{KE.KE}}$. After that, D it runs $\mathsf{Sim}_{\mathsf{R}}^{\mathsf{OT.Setup}}(1^\kappa, n_{\mathsf{ENC}})$, obtaining $\mathsf{pp}^{\mathsf{OT}}$ and the $\mathsf{view}_{\mathsf{R}}^{\pi^{\mathsf{OT.Setup}}}$, and it sets

$$\mathsf{view}_{\mathsf{R}}^{\mathsf{Setup}} := ((\mathsf{pp}^{\mathsf{KE}}, \mathsf{pp}^{\mathsf{OT}}, n_{\mathsf{ENC}}, \mathsf{F}), \mathsf{view}_{\mathsf{R}}^{\mathsf{KE.Setup}}, \mathsf{view}_{\mathsf{R}}^{\pi^{\mathsf{OT}}.\mathsf{Setup}}),$$

which it sends to A. When A outputs (M, r), D picks $k_r \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$, computes $z_r = \mathsf{F}_k(P_r)$, $y_r = z_r \oplus k_r$. Moreover, D computes $k_{r,E} = \mathsf{F}_{k_r}(P_2)$, and calls its oracle on input P_2 obtaining $k_{\bar{r},E}$. It also computes the ciphertexts $c_0 = \mathsf{Enc}_{k_{0,E}}(m_0)$, and $c_1 = \mathsf{Enc}_{k_{1,E}}(m_1)$. Finally, D calls its oracle on input $(\mathsf{pp}_{\mathsf{OT}}, r, y_r)$, obtaining the view $\mathsf{view}_{\mathsf{R}}^{\mathsf{OT}}$. At the end, D answers A the view

$$\mathsf{view}_{\mathsf{R}} = (\mathsf{view}_{\mathsf{R}}^{\mathsf{KE}}, (c_0, c_1), \mathsf{view}_{\mathsf{R}}^{\mathsf{OT}}).$$

When A outputs its bit b, D sees it, and D outputs as its output bit. D runs in time $t + t_{\mathsf{Sim,KE}} + t_{\mathsf{Sim,OT}} + t_{\mathsf{F}} + 2t_{\mathsf{Enc}} \leq t_2$ and does one query to F.

If the oracle D interacts with, outputs a real execution of F, D returns the H_{R}^2-view to A, otherwise the H_{R}^3-view (because $k_{\bar{r}}$ is a random value).

– **The t_3-adversary E against the Eav-secure encryption scheme ENC:** E has to distinguish whether it receives the encryption of $m_{\bar{r}}$ or of a random message $\tilde{m}_{\bar{r}}$.
At the start of the game, E receives 1^κ from A. It chooses $n_{\mathsf{ENC}}, \mathsf{F}, P_0, P_1, P_2$ as in the setup phase 1). Then, it runs $\mathsf{Sim}_{\mathsf{R}}^{\mathsf{KE}}(1^\kappa)$, receiving the public parameters $\mathsf{pp}_{\mathsf{KE}}$, the output k_0 and the view $\mathsf{view}_{\mathsf{R}}^{\mathsf{KE}}$. E splits it into $\mathsf{view}_{\mathsf{R}}^{\mathsf{KE.Setup}}$ and $\mathsf{view}_{\mathsf{R}}^{\mathsf{KE.KE}}$. After that, D it runs $\mathsf{Sim}_{\mathsf{R}}^{\mathsf{OT.Setup}}(1^\kappa, n_{\mathsf{ENC}})$, obtaining $\mathsf{pp}^{\mathsf{OT}}$ and the $\mathsf{view}_{\mathsf{R}}^{\pi^{\mathsf{OT.Setup}}}$, and it sets

$$\mathsf{view}_{\mathsf{R}}^{\mathsf{Setup}} := ((\mathsf{pp}^{\mathsf{KE}}, \mathsf{pp}^{\mathsf{OT}}, n_{\mathsf{ENC}}, \mathsf{F}), \mathsf{view}_{\mathsf{R}}^{\mathsf{KE.Setup}}, \mathsf{view}_{\mathsf{R}}^{\pi^{\mathsf{OT}}.\mathsf{Setup}}),$$

which it sends to A. When A outputs (M, r), E picks $k_r \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$, computes $z_r = \mathsf{F}_k(P_r)$, $y_r = z_r \oplus k_r$. Moreover, E computes $k_{r,E} = \mathsf{F}_{k_r}(P_2)$. It also computes the ciphertext $c_r = \mathsf{Enc}_{k_{r,E}}(m_r)$. Then, it calls its oracle on input $(m_{\bar{r}}, \tilde{m}_{\bar{r}})$, obtaining $c_{\bar{r}}$, with . Finally, D calls its oracle on input $(\mathsf{pp}_{\mathsf{OT}}, r, y_r)$, obtaining the view $\mathsf{view}_{\mathsf{R}}^{\mathsf{OT}}$. At the end, E answers A the view

$$\mathsf{view}_{\mathsf{R}} = (\mathsf{view}_{\mathsf{R}}^{\mathsf{KE}}, (c_0, c_1), \mathsf{view}_{\mathsf{R}}^{\mathsf{OT}}).$$

When A outputs its bit b, E sees it, and E outputs as its output bit. E runs in time $t + t_{\mathsf{Sim,KE}} + t_{\mathsf{Sim,OT}} + 2t_{\mathsf{F}} + t_{\mathsf{Enc}} \leq t_3$ and does one query to F.

If the oracle E interacts with, outputs the encryption of $m_{\bar{r}}$, E returns the H_{R}^3-view to A, otherwise the H_{R}^4-view since $k_{\bar{r},E}$ is random.

Algorithm 1 Our $\pi^{\mathsf{LR\text{-}OT}}$ protocol for a single OT instance.

```
– Building blocks:
    1. $\pi^{\mathsf{KE}} = (\pi^{\mathsf{KE}}.\mathsf{Setup}, \pi^{\mathsf{KE}}.\mathsf{KE})$, a key-exchange protocol,
    2. $\mathsf{F}$, a block-cipher,
    3. $\mathsf{ENC} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, an $\mathsf{EavL}$-secure encryption scheme,
    4. $\pi^{\mathsf{OT}} = (\pi^{\mathsf{OT}}.\mathsf{Setup}, \pi^{\mathsf{OT}}.\mathsf{OT})$, an $\mathsf{OT}$ protocol.
– Input: $\mathsf{S}$ has a couple of strings $M := (m_0, m_1)$, with $|m_0| = |m_1|$; $\mathsf{R}$ has a bit $r$
– Auxiliary input: $1^\kappa$ the security parameter (shared by both $\mathsf{S}$ and $\mathsf{R}$)
– Setup phase: $\pi^{\mathsf{LR\text{-}OT}}.\mathsf{Setup}(1^\kappa)$ does:
    1. From $1^\kappa$ choose $n_{\mathsf{ENC}}$, a $\mathsf{BC}$ $\mathsf{F} : \{0,1\}^{n_{\mathsf{ENC}}} \times \{0,1\}^{n_{\mathsf{ENC}}} \to \{0,1\}^{n_{\mathsf{ENC}}}$, an $\mathsf{EavL}$-
       secure encryption scheme $\mathsf{ENC}$, three different strings $P_0, P_1, P_2 \in \{0,1\}^{n_{\mathsf{ENC}}}$.
    2. $\mathsf{pp}^{\mathsf{KE}} \leftarrow \pi^{\mathsf{KE}}.\mathsf{Setup}(1^\kappa, n_{\mathsf{ENC}})$,
    3. $\mathsf{pp}^{\mathsf{OT}} \leftarrow \pi^{\mathsf{OT}}.\mathsf{Setup}(1^\kappa)$
    4. Return $\mathsf{pp} = (\mathsf{pp}^{\mathsf{KE}}, \mathsf{pp}^{\mathsf{OT}}, n_{\mathsf{ENC}}, \mathsf{F}, \mathsf{ENC}, P_0, P_1, P_2)$
– Main phase: $\pi^{\mathsf{LR\text{-}OT}}.\mathsf{OT}(\mathsf{pp})$:
    1. Key-exchange phase: $\mathsf{S}$ and $\mathsf{R}$ execute $\pi^{\mathsf{KE}}.\mathsf{KE}(\mathsf{pp}^{\mathsf{KE}})$ to both obtain $k$
    2. Sender's phase:
        (a) $\mathsf{S}$ picks $k_0, k_1 \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$
        (b) $\mathsf{S}$ computes $z_0 = \mathsf{F}_k(P_0),\ y_0 = z_0 \oplus k_0,\ z_1 = \mathsf{F}_k(P_1),\ y_1 = z_1 \oplus k_1$,
        (c) $\mathsf{S}$ computes $k_{0,E} = \mathsf{F}_{k_0}(P_2),\ k_{1,E} = \mathsf{F}_{k_1}(P_2)$,
        (d) $\mathsf{S}$ computes $c_0 = \mathsf{Enc}_{k_{0,E}}(m_0),\ c_1 = \mathsf{Enc}_{k_{1,E}}(m_1)$
        (e) $\mathsf{S}$ sends $c_0, c_1$ to $\mathsf{R}$
    3. Send-key phase: $\mathsf{S}$ and $\mathsf{R}$ execute $\pi^{\mathsf{OT}}.\mathsf{OT}(\mathsf{pp}^{\mathsf{OT}})$ with $\mathsf{S}$'s input $(y_0, y_1)$ and
       $\mathsf{R}$'s $r$
    4. Receiver's phase:
        $\mathsf{R}$ computes $k_{\mathsf{R}} = \mathsf{F}_k(P_r) \oplus y_r,\ k_{\mathsf{R},E} = \mathsf{F}_{k_{\mathsf{R}}}(P_2)$,
        $\mathsf{R}$ computes $m = \mathsf{Dec}_{k_{\mathsf{R},E}}(c_r)$
– Output: $\mathsf{S}$: nothing; $\mathsf{R}$: $m$
```

4.2 Security against Side-Channel Adversaries

We next consider security against side-channel adversaries. We start by imposing additional security requirements on the underlying building blocks. However, we emphasize that it is insufficient to model the security of each building block separately in the presence of leakage because an adversary can combine the leakage obtained from different building blocks using the same secret. For example, in the $\pi^{\mathsf{LR\text{-}OT}}$ protocol, the key k is obtained through the π^{KE} protocol and then used as a key by F. The following example illustrates the risks of combining leakages: Suppose that the π^{KE} leakage allows the adversary to recover all the even bits of the key, while the F leakage allows the adversary to recover all the odd bits. The key remains unpredictable with either leakage, but an adversary can recover the key with both. Therefore, we identify and model these leakage threats for composed sub-computations and design $\pi^{\mathsf{LR\text{-}OT}}$ while minimizing the usage of the same secret value. In particular, we use the encryption scheme twice, each time with a different key $k_{i,E}$, and F at most twice with the same key.

Security against intercepting side-channel adversaries:
Similarly to the natural paradigm discussed in Sec. 1.2, we require no leakage protection for the underlying π^{OT}, because the receiver is not corrupted. We will start by discussing how to enhance the building blocks and internal sub-computations against leakage (Steps denote listed steps in Alg. 1, App. A):

1. π^{KE} (Step 1): We require unpredictability (Def. 20) in the presence of leakage.
2. F (Steps 2b, 2c): We require unpredictability (Def. 6 where F is an ideal cipher).
3. \$-random sampling algorithm (Step 2a): We require unpredictability (Def. 17).
4. $\oplus$-XOR (Step 2b): We require unpredictability with a known output (Def. 18).
5. ENC (Step 2d): We require EavL security (Def. 8).
6. π^{OT} (Step 3): We require no security in the presence of leakage. That is, the leakage function may reveal the sender's input, y_0, and y_1.

We give the security definitions for our building blocks when they have not already been defined in the literature:

Unpredictability with leakage for random samplings This definition captures the fact that a SCA-adversary should not guess the output of a random sampling, even if it gets the leakage of the random sampling.

Definition 17. *Let \$ be the random sampling, and $\mathsf{L}_{\$}(x)$ be the leakage when the value picked is x. We say that the implementation of the random sampling is* (q_G, t, ϵ)-unpredictable with leakage $\mathsf{L}_{\$}$ *if for any t-adversary* A*,*

$$\Pr[k \in \mathcal{G} \mid |\mathcal{G}| \leq q_G,\ \mathcal{G} \leftarrow \mathsf{AL}_{\$}(k)),\ (k, \mathsf{L}_{\$}(k)) \xleftarrow{\$} \{0,1\}^n] \leq \epsilon,$$

Unpredictability with Leakage for the $\oplus$ function. With this definition, we want to capture the fact that a SCA-adversary cannot guess the inputs of the XOR function, even if it gets the leakage. We consider two cases: one in which, in addition to the leakage, the adversary obtains the output of the XOR function, and the other in which it chooses one of the inputs of the XOR function (it has to guess the other). This is the definition when the adversary gets the output of the XOR function:

Definition 18. *Let $\oplus$ be the XOR function and $\mathsf{L}_{\oplus}(\cdot,\cdot)$ be the leakage function of its implementation. We say that the implementation of the XOR function is* (q_G, t, ϵ)-unpredictable with leakage $\mathsf{L}_{\oplus}$ with known output *if for any t-adversary* A*,*

$$\Pr[k \in \mathcal{G} \mid |\mathcal{G}| \leq q_G,\ \mathcal{G} \leftarrow \mathsf{A}(y, \mathsf{L}_{\oplus}(z,k)),\ y = z \oplus k,\ z, k \xleftarrow{\$} \{0,1\}^n] \leq \epsilon.$$

This situation happens in the $\mathsf{OT\text{-}S\text{-}LA}$ case, since the adversary knows y_i via leakage ($y_i = z_i \oplus k_i$, with z_i being random as it is the output of $\mathsf{F}_k(P_i)$, for a random k).

Now, we give the security definition when the adversary chooses one of the inputs of the XOR function:

Definition 19. *Let $\oplus$ be the XOR function and $\mathsf{L}_{\oplus}(\cdot,\cdot)$ be the leakage function of its implementation. We say that the implementation of the XOR function is*

(q_G, t, ϵ)-unpredictable with leakage $\mathsf{L}_\oplus$ for chosen input *if for any t-adversary* $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$,

$$\Pr[k \in \mathcal{G} \mid |\mathcal{G}| \leq q_G,\ \mathcal{G} \leftarrow \mathsf{A}_2(\mathsf{st}, \mathsf{L}_\oplus(z,k)),\ y = z \oplus k,$$
$$k \overset{\$}{\leftarrow} \{0,1\}^n,\ (\mathsf{st}, z) \leftarrow \mathsf{A}_1] \leq \epsilon.$$

This situation happens in the $\mathsf{OT{-}S{-}LR}$ case, since the adversary knows z_i ($y_i = z_i \oplus k_i$) since the receiver knows k and z_i is the output of $\mathsf{F}_k(P_i)$.

Unpredictability with leakage for key-exchange protocols. In this definition we capture the fact that a SCA-adversary cannot guess the key output by a key-exchange protocol, even if it gets the leakage from the execution of the key-exchange protocol by one party. For ease of reading, we use the more compact notation $\mathsf{L}_{\mathsf{KE}}(k)$ to denote the leakage of the key-exchange protocol with input (1^κ) and randomness R when it outputs k.

Definition 20. *Let π^{KE} be a key-exchange protocol with output in $\{0,1\}^n$, and $\mathsf{L}_{\mathsf{KE}}(\cdot)$ a leakage function. We say that the implementation of π^{KE} is (q_G, ϵ)-*unpredictable with leakage L_{KE} *if for any t-adversary* A,

$$\Pr[k \in \mathcal{G} \mid |\mathcal{G}| \leq q_G,\ \mathcal{G} \leftarrow \mathsf{A}(\mathsf{pp}^{\mathsf{KE}}, \mathsf{trans}_{\mathsf{KE}}, \mathsf{L}^1_{\mathsf{KE}}(k)),$$
$$(k, \mathsf{pp}^{\mathsf{KE}}, \mathsf{trans}_{\mathsf{KE}}, \mathsf{L}_{\mathsf{KE}}(k)) \leftarrow \pi^{\mathsf{KE}}\mathsf{L}(1^\kappa, n)] \leq \epsilon.$$

Building blocks using the same secret Next, we identify the building blocks and sub-computations that use the same secret value:

1. π^{KE} and F: π^{KE} generates k (Step 1), while F uses k as a key (Step 2b);
2. \$, $\oplus$, and F: \$ generates k_i (Step 2a), $\oplus$ computes $z_i \oplus k_i$ (Step 2b), and F uses k_i as a key (Step 2c);
3. F and ENC: F generates $k_{i,E}$ as an output (Step 2c), and Enc uses as its key (Step 2d).

We provide new security definitions that capture these compositions and are inspired by q–upL (Def. 6), and EavL (Def. 8). We also discuss why these compositions can be considered minimal.

(1) π^{KE} *and* F*.* In our protocol, the sender executes π^{KE} to generate k, and then calls $\mathsf{F}_k(\cdot)$. Our security requirement is that the adversary cannot guess k even if it has the transcript and the leakage within protocol π^{KE} that generates k, and then an oracle access to $\mathsf{FL}_k(\cdot)$. This means that not only do we want both the implementations of π^{KE} and F to be unpredictable with leakage, but their composition must also be unpredictable. We cannot avoid using a key-exchange protocol because we need to establish a joint secret state, which must be used in the protocol by another cryptographic primitive. Moreover, a symmetric block-cipher is the simplest primitive that can be combined in terms of computation complexity (and therefore its associated leakage as well[5]). In this sense, this composition is minimal. Furthermore, an implementation idea to protect such a composition is to use masking: π^{KE} could output shares of the key that could

[5] It is well known that computation with asymmetric primitives leaks far more on each secret-bit of (say) the key as compared to symmetric primitives.

be used directly by the implementation of F. Our starting point is the q-upL definition (Def. 6) where we replace the random sampling of the key with the key-exchange protocol so that the adversary gets the transcript of π^{KE} and the leakage of the sender executing it. Formally,

Definition 21. *Let π^{KE} be a key-exchange protocol with output in $\{0,1\}^n$, and $\mathsf{L}_{\mathsf{KE}}(\cdot)$ be a leakage function. Let $\mathsf{F} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be an ideal cipher and L_{F} be a leakage function (Sec. 2.2). Let π^{KE} generate the key k used by oracle FL. We say that the implementation of the composed computation of π^{KE} and F on k is $(q, q_G, q_{\mathsf{F}}, t, \epsilon)$-*unpredictable with leakage $(\mathsf{L}_{\mathsf{KE}}, \mathsf{L}_{\mathsf{F}})$*, denoted as q-upL-KE, if for any (q, q_{F}, t)-adversary A,*

$$\Pr[k \in \mathcal{G} \mid |\mathcal{G}| \le q_G,\ \mathcal{G} \leftarrow \mathsf{A}^{\mathsf{FL}_k(\cdot),\mathsf{F}.(\cdot)}(\mathsf{pp}^{\mathsf{KE}}, \mathsf{trans}_{\mathsf{KE}}, \mathsf{L}^1_{\mathsf{KE}}(k)),$$
$$(k, \mathsf{pp}^{\mathsf{KE}}, \mathsf{trans}_{\mathsf{KE}}, \mathsf{L}_{\mathsf{KE}}(k)) \leftarrow \pi^{\mathsf{KE}}\mathsf{L}(1^\kappa, n)] \le \epsilon,$$

with q_{F} and the oracle FL defined as in Def. 6.

This formalization forces the adversary first to receive the leakage from π^{KE} and then query FL_k as required by $\pi^{\mathsf{LR\text{-}OT}}$. Looking ahead, our theorem requires $q = 2$.

(2) *\$, $\oplus$ and* F*.* In our protocol, S picks k_i, computes $y_i = z_i \oplus k_i$, where z_i a random value (since it is the output of an ideal cipher), and uses it once as a key of F. We need to encrypt the key of F, and we do this in the simplest way: XORing it with a random value. Note that the XOR function is the simplest function regarding computational cost (and leakage). Moreover, we can easily and cheaply protect it with masking owing to its linearity and, therefore, the linear cost of masking it [16, 52]). We believe this requirement is minimal because we are using a secret that may or may not be known to the receiver (it depends on the receiver's input). Thus, we have its generation (random sampling), its use (as a key for F), and its encryption (XORing with a random value). In every case, we have used the simplest possible functions. As before, our starting point is the q-upL definition[6], where we add the random sampling leakage and the output of the XOR of k with a random value, z. Finally, we give the leakage of the generation of z, $\mathsf{L}^{out}_{\mathsf{F}}(k'; z)$. Since z is obtained as the output of F, according to [9], we give to the adversary $\mathsf{L}^{out}_{\mathsf{F}}(k'; z)$ for a key k' chosen by the adversary. The adversarial choice of k' models the fact that k' (k in Alg. 1) is picked independently of the three components we combine here, so we consider the worst possible scenario: k' known and chosen by the adversary. Formally,

Definition 22. *Let \$ be the random sampling, $\oplus$ be the XOR function, and $\mathsf{L}_{\$}(x)$ and $\mathsf{L}_{\oplus}(x, y)$ be the leakage of their respective implementations. Let $\mathsf{F} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be an ideal cipher and L_{F} a leakage function. Let \$ generate k, used by the oracle FL. Let k be XORed to a random value z obtained by F. We say that the implementation of the composition of the random sampling,*

[6] We are also inspired by the definition of security in the presence of leakage of the XOR of a pseudorandom value with a message block [9], where we give the leakage of the generation of the random block.

the XOR and F *is* $(q, q_G, q_{\mathsf{F}}, t, \epsilon)$-unpredictable with leakage $(\mathsf{L}_\$, \mathsf{L}_\oplus, \mathsf{L}_\mathsf{F})$*, denoted as* q-upL-$\mathsf{\$XORL}$*, if for any* (q, q_F, t)*-adversary* A*,*

$$\Pr[k \in \mathcal{G} \mid |\mathcal{G}| \leq q_G,\ \mathcal{G} \leftarrow \mathsf{A}_2^{\mathsf{FL}_k(\cdot),\mathsf{F}.(\cdot)}(\mathsf{st}, \mathsf{L}_\$(k), y, \mathsf{L}_\oplus(z,k), \mathsf{L}_\mathsf{F}^{out}(k';z)),$$
$$y = z \oplus k, (k, \mathsf{L}_\$(k)) \xleftarrow{\$} \{0,1\}^n, z \xleftarrow{\$} \{0,1\}^n,\ (\mathsf{st}, k') \leftarrow \mathsf{A}_1^{\mathsf{F}.(\cdot)}] \leq \epsilon,$$

with q_F *and the oracle* FL *defined as in Def. 6.*

Looking ahead, we will use this security definition twice: once to require that k_0 is not guessed and once for k_1. This formalization is coherent with the $\pi^{\mathsf{LR\text{-}OT}}$ flow. In our case, $q = 1$ or 2.

(3) F *and* ENC. In contrast to the previous two compositions, we move to a definition of indistinguishability. The reason is that we are combining an encryption scheme with its key generation (where we generate the key using F, which is a simple cryptographic primitive). Therefore, we require that even if the adversary gets the key-generation leakage of ENC, it cannot distinguish the encryption of two messages. So we simply modify Def. 8 by adding the key-generation leakage $\mathsf{L}_\mathsf{F}^{out}(k_i'; k_{i,E})$ for a k_i' chosen by the adversary. Formally,

Definition 23. *Let* $\mathsf{F} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ *be an ideal cipher and* L_F *be a leakage function. Let* ENC *be an encryption scheme (Def. 7) and* $\mathsf{L}_{\mathsf{Enc}}$ *be the leakage function of its encryption algorithm. Let* F *generate the key of* ENC*. The implementation of the composition is* $(q_\mathsf{L}, q_E, q_\mathsf{F}, t, \epsilon)$-$\mathsf{CPAL}$-$\mathsf{GenFL}$-secure in the presence of leakage when the key picked via FL*, if for any* (q_E, q_F, t)*-adversary* $\mathsf{A} = (\mathsf{A}_0, \mathsf{A}_1, \mathsf{A}_2)$

$$\Big| \Pr[b = b' \mid b' \leftarrow \mathsf{A}_2^{\mathsf{EncL}_k(\cdot),\mathsf{F}.(\cdot)}(c^*, \ell^*, \mathsf{st}),\ (c^*, \ell^*) = \mathsf{EncL}_k(m_b),$$
$$(\mathsf{st}, m_0, m_1) \leftarrow \mathsf{A}_1^{\mathsf{EncL}_k(\cdot),\mathsf{F}.(\cdot)}(\mathsf{st}, \mathsf{L}_\mathsf{F}^{out}(k';k))\ s.t.\ |m_0| = |m_1|,$$
$$b \xleftarrow{\$} \{0,1\}, k \xleftarrow{\$} \{0,1\}^n, (\mathsf{st}, k') \leftarrow \mathsf{A}_0^{\mathsf{F}.(\cdot)}] - \frac{1}{2} \Big| \leq \epsilon$$

If $q_E = 0$*, we say that* Π *is* (q_E, t, ϵ)-Eav-GenFL-secure.

Since $k_{i,E} = \mathsf{F}_{k_i}(P_2)$ can be seen as a refreshed key of ENC, if ENC is a EavL-secure scheme that uses F to refresh its key, then ENC is a EavL-GenFL-secure encryption scheme for free [45, 9].

OT-S-LA-security of $\pi^{\mathsf{LR\text{-}OT}}$. We are ready to state and prove the OT-S-LA-security of $\pi^{\mathsf{LR\text{-}OT}}$,

Theorem 2. *Let* $\mathsf{F} : \{0,1\}^{n_{\mathsf{ENC}}} \times \{0,1\}^{n_{\mathsf{ENC}}} \to \{0,1\}^{n_{\mathsf{ENC}}}$ *be an ideal blockcipher. Let* F *be* $(2, q_I + 2, q_I + 2, t_1, \epsilon_{2\text{-}\mathsf{upL}\text{-}\mathsf{KE}})$-2-$\mathsf{upL}$-$\mathsf{KE}$*-secure with the key generated by* π^{KE}*, and be* $(1, q_I + 2, q_I + 2, t_2, \epsilon_{1\text{-}\mathsf{upL}\text{-}\mathsf{\$XORL}})$-1-$\mathsf{upL}$-$\mathsf{\$XORL}$*-secure with the key randomly sampled and XORed to a random value. Let* ENC *be* $(q_I + 3, t_3, \epsilon_{\mathsf{EavL}\text{-}\mathsf{GenFL}})$-$\mathsf{EavL}$-$\mathsf{GenFL}$ *with the key picked via* F*, then, for any* (q_F, t) *adversary* $\mathsf{A} = (\mathsf{A}_1^\mathsf{F}, \mathsf{A}_2^\mathsf{F})$*-adversary,* $\pi^{\mathsf{LR\text{-}OT}}$ *is* $(q_\mathsf{F}, t, \epsilon)$-$\mathsf{OT}$-$\mathsf{S}$-$\mathsf{LA}$ *secure, with*

$$\epsilon \leq \epsilon_{2\text{-}\mathsf{upL}\text{-}\mathsf{KE}} + 2(\epsilon_{1\text{-}\mathsf{upL}\text{-}\mathsf{\$XORL}} + 2\epsilon_{\mathsf{EavL}\text{-}\mathsf{GenFL}}).$$

$t_1 = t + t_{\mathsf{OT}} + 2t_{\mathsf{Enc}}$*,* $t_2 = t + t_{\mathsf{KE}} + t_{\mathsf{OT}} + 2t_{\mathsf{Enc}}$*,* $t_3 = t + t_{\mathsf{KE}} + t_{\mathsf{OT}} + t_{\mathsf{Enc}}$*.*

Proof Idea: Our adversary has to distinguish whether the sender is sending (m_0^0, m_1^0) or (m_0^1, m_1^1) (Def. 15). Roughly speaking, if the adversary cannot guess k (2-upL-KE), both z_0 and z_1 remain random.[7] Thus, an adversary that knows y_0 and y_1, cannot guess the keys k_i's (1-upL-\$XORL), so, the keys $k_{i,E}$'s remain random. Relying on the EavL-GenFL-security of ENC, an adversary cannot distinguish whether c_i is an encryption of m_i^0 or m_i^1.

Proof. The proof is inspired by those in the same model for F in [9].

Hybrids. We use a sequence of Hybrids, Hybrid 0, ..., and Hybrid 8, with an adversary A. Let E_i be the event that the adversary A outputs 1 at the end of Hybrid i.

- **Hybrid 0:** It is the OT-S-LA Hybrid where A is playing against $\pi^{\mathsf{LR\text{-}OT}}$, where the bit picked is 0.
- **Hybrid 1:** It is Hybrid 0, where we replace z_0, z_1 with random values, and the leakage of F accordingly.
- **Hybrid 2:** It is Hybrid 1, where we replace $y_0, k_{0,E}$ with two random values and their leakage accordingly.
- **Hybrid 3:** It is Hybrid 2, where we replace $y_1, k_{1,E}$ with two random values and their leakage accordingly.
- **Hybrid 4:** It is Hybrid 3, where we replace c_0 with $\mathsf{Enc}_{k_{0,E}}(m_0^1)$ (that is, instead of encrypting m_0^0, we encrypt m_0^1), and its leakage accordingly.
- **Hybrid 5:** It is Hybrid 4, where we replace c_1 with $\mathsf{Enc}_{k_{1,E}}(m_1^1)$ (that is, instead of encrypting m_1^0, we encrypt m_1^1), and its leakage accordingly.
- **Hybrid 6 (Complementing 3):** It is Hybrid 5, where we replace y_1 and $k_{1,E}$ with the correct values and the leakage accordingly.
- **Hybrid 7 (Complementing 2):** It is Hybrid 6, where we replace y_0 and $k_{0,E}$ with the correct values and the leakage accordingly.
- **Hybrid 8 (Complementing 1):** It is Hybrid 7, where we replace z_0, z_1 with the correct values, and the leakage accordingly.

Indistinguishability of Hybrids. Each of the previous Hybrids is indistinguishable from the following, as we now prove:

- **From Hybrid 0 to 1:** We observe the two Hybrids are the same except if the adversary A does a query to F with the key k because it would cause the loss of randomness of the ys [59]. Thus,
$$|\Pr[E_0] - \Pr[E_1]| \leq \epsilon_{\mathsf{2\text{-}upL\text{-}KE}}$$
- **From Hybrid 1 to 2:** We observe the two Hybrids are the same except if the adversary A does a query to F with the key k_0 because it would cause the loss of randomness of y_0 or $k_{0,E}$. Thus,
$$|\Pr[E_1] - \Pr[E_2]| \leq \epsilon_{\mathsf{1\text{-}upL\text{-}\$XORL}}$$
- **From Hybrid 2 to 3:** We observe the two Hybrids are the same except if the adversary A does a query to F with the key k_1 because it would cause the loss of randomness of y_1 or $k_{1,E}$. Thus,
$$|\Pr[E_2] - \Pr[E_3]| \leq \epsilon_{\mathsf{1\text{-}upL\text{-}\$XORL}}$$

[7] If k is unpredictable, then passing through an ideal cipher gives random outputs.

- **From Hybrid 3 to 4:** Since ENC is $(q_{3,\mathsf{F}}, t_3, \epsilon_{\mathsf{EavL-GenFL}})$ and the result of Lemma 1
$$|\Pr[E_3] - \Pr[E_4]| \leq 2\epsilon_{\mathsf{EavL-GenFL}}$$
- **From Hybrid 4 to 5:** Similarly, as before
$$|\Pr[E_4] - \Pr[E_5]| \leq 2\epsilon_{\mathsf{EavL-GenFL}}$$
- **From Hybrid 5 to 8:** It is the indistinguishability of Hybrids 0, 1, 2, and 3 done in the opposite order. Thus,
$$|\Pr[E_5] - \Pr[E_8]| \leq \epsilon_{\mathsf{2-upL-KE}} + 2\epsilon_{\mathsf{1-upL-\$XORL}}$$

Now, we observe that Hybrid 0 is the $\mathsf{OT-S-LA}$ game where the bit picked is 0, while Hybrid 8 is the $\mathsf{OT-S-LA}$ game where the bit is picked 1. Thus,
$$|\Pr[E_0] - \Pr[E_8]| \leq 4\epsilon_{\mathsf{EavL-GenFL}} + 4\epsilon_{\mathsf{1-upL-\$XORL}} + 2\epsilon_{\mathsf{2-upL-KE}}.$$
Using Lemma 1
$|\Pr[\mathsf{A}\text{ wins}]-1/2| \leq 1/2(\Pr[E_0]-\Pr[E_8]) \leq \epsilon_{\mathsf{2-upL-KE}}+2\epsilon_{\mathsf{1-upL-\$XORL}}+2\epsilon_{\mathsf{EavL-GenFL}}$.
Adversaries used in the hybrids' transitions. To finish the proof, we have to explicitly describe the adversaries used in the reduction between the hybrids:

- **The** $(2, q_G, q_{1,F}, t_1)$**-2-upL-KE-adversary** B **against the composition of** π^{KE} **and** F**:** B has to guess the key k produced by π^{KE} and used twice by F. At the start of the game, B has oracle access to F and receives 1^κ from A. It chooses $n_{\mathsf{ENC}}, \mathsf{ENC}, P_0, P_1, P_2$ as in the setup phase 1) and π^{OT}. Then, B obtains $\mathsf{pp_{KE}}$, the transaction $\mathsf{trans}^{\mathsf{KE.Setup}}$, and the leakage $\ell^{\mathsf{S}}_{\mathsf{KE.Setup}}(1^\kappa)$ from its oracle. After that, B run the $\pi^{\mathsf{OT.Setup}}$ sub-protocol on input $(1^\kappa, n_{\mathsf{ENC}})$, obtaining $\mathsf{pp}^{\mathsf{OT}}$, the transaction $\mathsf{trans}^{\pi^{\mathsf{OT}}.\mathsf{Setup}}(1^\kappa, n_{\mathsf{ENC}})$ and the leakage $\ell^{\mathsf{S}}_{\mathsf{OT.Setup}}$. It sets
$$\mathsf{trans}^{\mathsf{Setup}} := ((\mathsf{pp}^{\mathsf{KE}}, \mathsf{pp}^{\mathsf{OT}}, n_{\mathsf{ENC}}, \mathsf{F}), \mathsf{trans}^{\mathsf{KE.Setup}}, \mathsf{trans}^{\pi^{\mathsf{OT}}.\mathsf{Setup}}),$$
$$\text{and the leakage } \ell^{\mathsf{S}}_{\mathsf{Setup}} := (\ell^{\mathsf{S}}_{\mathsf{KE.Setup}}, \ell^{\mathsf{S}}_{\mathsf{OT.Setup}}),$$
which it sends to A.
When A outputs (M, r), B obtains from its oracle the leakage $\ell^{\mathsf{S}}_{\mathsf{KE}}(k)$, and the transaction $\mathsf{trans_{KE}}$. Then, it picks $k_0, k_1 \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$, collecting the leakages $\mathsf{L}_\$(k_0)$, and $\mathsf{L}_\$(k_1)$. After that, B calls its oracle FL_k on input P_0 and P_1, obtaining z_0, z_1, and the leakages $\mathsf{L_F}(k; P_0)$, and $\mathsf{L_F}(k; P_1)$. Then, it computes $y_0 = z_0 \oplus k_0$, $y_1 = z_1 \oplus k_1$, and collects the leakages $\mathsf{L}_\oplus(z_0, k_0)$ and $\mathsf{L}_\oplus(z_1, k_1)$. Moreover, B calls the ideal cipher on input (k_0, P_2) and (k_1, P_2), obtaining the encryption keys $k_{0,E}$, $k_{1,E}$, and the leakages $\mathsf{L_F}(k_0; P_2)$, $\mathsf{L_F}(k_1; P_2)$, respectively. After that, it computes the ciphertexts $c_0 = \mathsf{Enc}_{k_{0,E}}(m^0_0)$, and $c_1 = \mathsf{Enc}_{k_{1,E}}(m^0_1)$ and their respective leakages $\mathsf{L_{Enc}}(k_{0,E}; m^0_0)$ $\mathsf{L_{Enc}}(k_{1,E}; m^0_1)$. Finally, B executes the $\pi^{\mathsf{OT.OT}}$ sub-protocol where the sender has input (y_0, y_1), and the receiver has input r, obtaining the transaction $\mathsf{trans}^{\mathsf{OT}}_{\mathsf{S}}$ and the leakage $\ell^{\mathsf{S}}_{\mathsf{OT.OT}}((y_0, y_1), r)$. At the end, B answers A the transaction
$$\mathsf{trans} = (\mathsf{trans}^{\mathsf{KE}}, (c_0, c_1), \mathsf{trans}^{\mathsf{OT}}), \text{and the leakage}$$
$$\ell^{\mathsf{S}} = (\ell^{\mathsf{S}}_{\mathsf{KE}}(k), \mathsf{L}_\$(k_0), \mathsf{L}_\$(k_1), \mathsf{L_F}(k; P_0), \mathsf{L_F}(k; P_1), \mathsf{L}_\oplus(z_0, k_0), \mathsf{L}_\oplus(z_1, k_1),$$
$$\mathsf{L_F}(k_0; P_2), \mathsf{L_F}(k_1; P_2), \mathsf{L_{Enc}}(k_{0,E}; m^0_0), \mathsf{L_{Enc}}(k_{1,E}; m^0_1), \ell^{\mathsf{S}}_{\mathsf{OT.OT}}((y_0, y_1), r)).$$

When A does one of its q_I ideal queries to its oracle, B does the same query. When A outputs its bit b, B outputs all the keys it has used in an oracle queries to F. These are at most $q_I + 2$. B runs in time $t + t_{\mathsf{OT}} + 2t_{\mathsf{Enc}} \leq t_1$, does $q_I + 2$ ideal queries.

If the adversary has not queried F with k, we can replace z_0 and z_1 with random values.

– **The** $(1, q_I + 2, q_I + 2, t_2)$**-1-upL-\$XORL-adversary C against the composition of \$, $\oplus$ and F:** C has to guess the key k_0 randomly picked, used once by F and XORed once to a random value.
At the start of the game, C has oracle access to F and receives 1^κ from A. It chooses $n_{\mathsf{ENC}}, \mathsf{ENC}, P_0, P_1, P_2$ as in the setup phase 1) and π^{OT}. Then, C performs $\pi^{\mathsf{KE.Setup}}(1^\kappa)$, obtaining $\mathsf{pp}_{\mathsf{KE}}$, the transaction $\mathsf{trans}^{\mathsf{KE.Setup}}$, and the leakage $\ell^{\mathsf{S}}_{\mathsf{KE.Setup}}(1^\kappa)$. After that, C run the $\pi^{\mathsf{OT.Setup}}(1^\kappa, n_{\mathsf{ENC}})$ sub-protocol on input $(1^\kappa, n_{\mathsf{ENC}})$, obtaining $\mathsf{pp}^{\mathsf{OT}}$, the transaction $\mathsf{trans}^{\pi^{\mathsf{OT}}.\mathsf{Setup}}(1^\kappa, n_{\mathsf{ENC}})$ and the leakage $\ell^{\mathsf{S}}_{\mathsf{OT.Setup}}$. It sets

$$\mathsf{trans}^{\mathsf{Setup}} := ((\mathsf{pp}^{\mathsf{KE}}, \mathsf{pp}^{\mathsf{OT}}, n_{\mathsf{ENC}}, \mathsf{F}), \mathsf{trans}^{\mathsf{KE.Setup}}, \mathsf{trans}^{\pi^{\mathsf{OT}}.\mathsf{Setup}}),$$
$$\text{and the leakage } \ell^{\mathsf{S}}_{\mathsf{Setup}} := (\ell^{\mathsf{S}}_{\mathsf{KE.Setup}}, \ell^{\mathsf{S}}_{\mathsf{OT.Setup}}),$$

which it sends to A.
When A outputs (M, r), C runs $\pi^{\mathsf{KE.KE}}(\mathsf{pp}_{\mathsf{KE}})$, obtaining k, the leakage $\ell^{\mathsf{S}}_{\mathsf{KE}}(k)$, and the transaction $\mathsf{trans}_{\mathsf{KE}}$. C outputs k and obtains $(\mathsf{L}_\$(k_0), y_0, \mathsf{L}_\oplus(z_0, k_0), \mathsf{L}^{out}_{\mathsf{F}}(k_0; z_0))$. It computes $\mathsf{L}^{in}_{\mathsf{F}}(k; P_0)$, and its sets $\mathsf{L}_{\mathsf{F}}(k; P_0) := (\mathsf{L}^{in}_{\mathsf{F}}(k; P_0), \mathsf{L}^{out}_{\mathsf{F}}(k_0; z_0))$. Then, it picks $k_1 \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$, collecting the leakage $\mathsf{L}_\$(k_1)$. After that, C calls F on input (k, P_1), obtaining z_1, and the leakage $\mathsf{L}_{\mathsf{F}}(k; P_1)$. Then, it computes $y_1 = z_1 \oplus k_1$, and collects the leakage $\mathsf{L}_\oplus(z_1, k_1)$. After that it calls its oracle FL_k on input (P_2), obtaining the encryption key $k_{0,E}$, and the leakages $\mathsf{L}_{\mathsf{F}}(P_2; k_0)$. Moreover, C calls F on input (k_1, P_2), obtaining the encryption key $k_{1,E}$, and the leakage $\mathsf{L}_{\mathsf{F}}(P_2; k_1)$. After that, it computes the ciphertexts $c_0 = \mathsf{Enc}_{k_{0,E}}(m^0_0)$, and $c_1 = \mathsf{Enc}_{k_{1,E}}(m^0_1)$ and their respective leakages $\mathsf{L}_{\mathsf{Enc}}(m^0_0; k_{0,E})$ $\mathsf{L}_{\mathsf{Enc}}(m^0_1; k_{1,E})$. Finally, C executes the $\mathsf{OT.OT}$ sub-protocol where the sender has input (y_0, y_1), and the receiver has input r, obtaining the transaction $\mathsf{trans}^{\mathsf{OT}}_{\mathsf{S}}$ and the leakage $\ell^{\mathsf{S}}_{\mathsf{OT.OT}}((y_0, y_1), r)$. At the end, C answers A the transaction

$$\mathsf{trans} = (\mathsf{trans}^{\mathsf{KE}}, (c_0, c_1), \mathsf{trans}^{\mathsf{OT}}), \text{and the leakage}$$
$$\ell^{\mathsf{S}} = (\ell^{\mathsf{S}}_{\mathsf{KE}}(k), \mathsf{L}_\$(k_0), \mathsf{L}_\$(k_1), \mathsf{L}_{\mathsf{F}}(k; P_0), \mathsf{L}_{\mathsf{F}}(k; P_1), \mathsf{L}_\oplus(z_0, k_0), \mathsf{L}_\oplus(z_1, k_1),$$
$$\mathsf{L}_{\mathsf{F}}(k_0; P_2), \mathsf{L}_{\mathsf{F}}(k_1; P_2), \mathsf{L}_{\mathsf{Enc}}(k_{0,E}; m^0_0), \mathsf{L}_{\mathsf{Enc}}(m_1; k_{1,E}), \ell^{\mathsf{S}}_{\mathsf{OT.OT}}((y_0, y_1), r)).$$

When A does one of its q_I ideal queries to its oracle, C does the same query. When A outputs its bit b, C outputs all the keys it has used in an oracle queries to F. These are at most $q_I + 2$. C runs in time $t + t_{\mathsf{OT}} + 2t_{\mathsf{Enc}} \leq t_2$, does $q_I + 2$ ideal queries and one to F_{k_0}.

If the adversary has not queried F with k_0, we can replace $k_{0,E}$ a with random value.

– We can use the same adversary replacing k_0 with k_1, y_0 with y_1, etc to prove that if the adversary has not queried F with k_1, we can replace $k_{1,E}$ a with random value.

– **The** $(q_I + 3, t_3)$**-$\mathsf{EavL\text{-}GenFL}$-adversary D against the composition of ENC and F:** D has to guess if $\mathsf{Enc}_{k_{0,E}}$ has encrypted m_0^0 or m_0^1, with $k_{0,E}$ obtained by F.
  At the start of the game, D has oracle access to F and receives 1^κ from A. It chooses $n_{\mathsf{ENC}}, \mathsf{ENC}, P_0, P_1, P_2$ as in the setup phase 1) and π^{OT}. Then, C performs $\pi^{\mathsf{KE.Setup}}(1^\kappa)$, obtaining $\mathsf{pp}_{\mathsf{KE}}$, the transaction $\mathsf{trans}^{\mathsf{KE.Setup}}$, and the leakage $\ell^{\mathsf{S}}_{\mathsf{KE.Setup}}(1^\kappa)$. After that, D run the $\pi^{\mathsf{OT.Setup}}(1^\kappa, n_{\mathsf{ENC}})$ sub-protocol on input $(1^\kappa, n_{\mathsf{ENC}})$, obtaining $\mathsf{pp}^{\mathsf{OT}}$, the transaction $\mathsf{trans}^{\pi^{\mathsf{OT}}.\mathsf{Setup}}(1^\kappa, n_{\mathsf{ENC}})$ and the leakage $\ell^{\mathsf{S}}_{\mathsf{OT.Setup}}$. It sets

$$\mathsf{trans}^{\mathsf{Setup}} := ((\mathsf{pp}^{\mathsf{KE}}, \mathsf{pp}^{\mathsf{OT}}, n_{\mathsf{ENC}}, \mathsf{F}), \mathsf{trans}^{\mathsf{KE.Setup}}, \mathsf{trans}^{\pi^{\mathsf{OT}}.\mathsf{Setup}}),$$
$$\text{and the leakage } \ell^{\mathsf{S}}_{\mathsf{Setup}} := (\ell^{\mathsf{S}}_{\mathsf{KE.Setup}}, \ell^{\mathsf{S}}_{\mathsf{OT.Setup}}),$$

  which it sends to A.
  When A outputs (M, r), D runs $\pi^{\mathsf{KE.KE}}(\mathsf{pp}_{\mathsf{KE}})$, obtaining k, the leakage $\ell^{\mathsf{S}}_{\mathsf{KE}}(k)$, and the transaction $\mathsf{trans}_{\mathsf{KE}}$. Then, it picks $k_0, k_1 \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$, collecting the leakages $\mathsf{L}_{\$}(k_0)$, and $\mathsf{L}_{\$}(k_1)$. After that, D calls F on input (k, P_0), and (k, P_1), obtaining z_0, and, z_1, and the leakages $\mathsf{L}_{\mathsf{F}}(k; P_0)$, and $\mathsf{L}_{\mathsf{F}}(k; P_1)$. Then, it computes $y_0 = z_0 \oplus k_0$, $y_1 = z_1 \oplus k_1$, and collects the leakages $\mathsf{L}_{\oplus}(z_0, k_0)$ and $\mathsf{L}_{\oplus}(z_1, k_1)$. Then D outputs k_0 and she obtains the leakage $\mathsf{L}^{out}_{\mathsf{F}}(k_0; k_{0,E})$. It computes $\mathsf{L}^{in}_{\mathsf{F}}(k_0; P_2)$ and she sets $\mathsf{L}_{\mathsf{F}}(k_0; P_2) := (\mathsf{L}^{in}_{\mathsf{F}}(k_0; P_2), \mathsf{L}^{out}_{\mathsf{F}}(k_0; k_{0,E}))$ After having received this, D outputs (m_0^0, m_0^1), and she obtains c_0^* and the leakage ℓ^*. Moreover, D calls the ideal cipher on input (k_1, P_2), obtaining the encryption key $k_{1,E}$, and the leakage $\mathsf{L}_{\mathsf{F}}(k_1; P_2)$. After that, it computes the ciphertext $c_1 = \mathsf{Enc}_{k_{1,E}}(m_1^0)$ and the leakage $\mathsf{L}_{\mathsf{Enc}}(m_1^0; k_{1,E})$. Finally, D executes the $\mathsf{OT.OT}$ sub-protocol where the sender has input (y_0, y_1), and the receiver has input r, obtaining the transaction $\mathsf{trans}^{\mathsf{OT}}_{\mathsf{S}}$ and the leakage $\ell^{\mathsf{S}}_{\mathsf{OT.OT}}((y_0, y_1), r)$. At the end, D answers A the transaction

$$\mathsf{trans} = (\mathsf{trans}^{\mathsf{KE}}, (c_0^*, c_1), \mathsf{trans}^{\mathsf{OT}}), \text{and the leakage}$$
$$\ell^{\mathsf{S}} = (\ell^{\mathsf{S}}_{\mathsf{KE}}(k), \mathsf{L}_{\$}(k_0), \mathsf{L}_{\$}(k_1), \mathsf{L}_{\mathsf{F}}(k; P_0), \mathsf{L}_{\mathsf{F}}(k; P_1), \mathsf{L}_{\oplus}(z_0, k_0), \mathsf{L}_{\oplus}(z_1, k_1),$$
$$\mathsf{L}_{\mathsf{F}}(k_0; P_2), \mathsf{L}_{\mathsf{F}}(k_1; P_2), \ell^*, \mathsf{L}_{\mathsf{Enc}}(m_1; k_{1,E}), \ell^{\mathsf{S}}_{\mathsf{OT.OT}}((y_0, y_1), r)).$$

  When A does one of its q_I ideal queries to its oracle, D does the same query. When A outputs its bit b, D output this as its output bit. These are at most $q_I + 3$ queries to F, since when A does an ideal cipher query, D does the same. D runs in time $t + t_{\mathsf{KE}} + t_{\mathsf{OT}} + t_{\mathsf{Enc}} \leq t_2$.

  If the bit b the game has picked is 0 D simulates correctly Hybrid 4, otherwise 5.

– We can use the same adversary replacing $k_{0,E}$ with $k_{1,E}$, c_0^* with c_1^*, etc to prove that the encryption of m_1^0 and m_1^1 are indistinguishable.

□

In the previous proof, we have used the following well-known lemma:

Lemma 1. *Let* ENC *be a* (ϵ)-CPA *encryption scheme. Then*

$$|\Pr[1 \leftarrow \mathsf{A}^{\mathsf{CPA}^0_{\mathsf{ENC}}}] - \Pr[1 \leftarrow \mathsf{A}^{\mathsf{CPA}^1_{\mathsf{ENC}}}]| \leq 2\epsilon,$$

where with $b' \leftarrow \mathsf{A}^{\mathsf{CPA}^b_{\mathsf{ENC}}}$ *we denote that* A *outputs* b' *at the end of the* CPA *game, where the encryption scheme is* ENC *and the bit picked is* b.

Proof.

$$\Pr[\mathsf{A}\text{ wins}] = (1 - \Pr[1 \leftarrow \mathsf{A}^{\mathsf{CPA}^0_{\mathsf{ENC}}}])\Pr[b=0] + \Pr[1 \leftarrow \mathsf{A}^{\mathsf{CPA}^1_{\mathsf{ENC}}}]\Pr[b=1] =$$
$$1/2 + 1/2(\Pr[1 \leftarrow \mathsf{A}^{\mathsf{CPA}^1_{\mathsf{ENC}}}] - \Pr[1 \leftarrow \mathsf{A}^{\mathsf{CPA}^0_{\mathsf{ENC}}}]).$$
$$\text{But } \ |1/2 + 1/2(\Pr[1 \leftarrow \mathsf{A}^{\mathsf{CPA}^1_{\mathsf{ENC}}}] - \Pr[1 \leftarrow \mathsf{A}^{\mathsf{CPA}^0_{\mathsf{ENC}}}] - 1/2| \leq \epsilon.$$
$$\text{Thus } \ 1/2|\Pr[1 \leftarrow \mathsf{A}^{\mathsf{CPA}^1_{\mathsf{ENC}}}] - \Pr[1 \leftarrow \mathsf{A}^{\mathsf{CPA}^0_{\mathsf{ENC}}}| \leq \epsilon,$$
$$\text{Thus } \ |\Pr[1 \leftarrow \mathsf{A}^{\mathsf{CPA}^1_{\mathsf{ENC}}}] - \Pr[1 \leftarrow \mathsf{A}^{\mathsf{CPA}^0_{\mathsf{ENC}}}| \leq 2\epsilon.$$

□

Security against corrupted side-channel receivers:

We continue with a corrupted side-channel receiver. The main difference with respect to an intercepting side-channel adversary is that when the adversary can corrupt the receiver, it knows the key k. Thus, we must require stronger security properties from the underlying π^{OT} protocol. In more detail, the differences between the security level of the underlying building blocks for OT–S–LR and OT–S–LA are as follows (Steps denote listed steps in Alg. 1, App. A):

- π^{KE} (Step 1): Since the receiver is corrupted, the adversary already knows the key k. Therefore, we require no security.
- $\oplus$ function (Step 2b): we require unpredictability (Def. 19). Unlike the previous setting, the adversary knows one of the inputs, namely, $z_{\bar{r}}$. We require the output and the remaining input to remain unpredictable.
- π^{OT} (Step 3): we require unpredictability of the sender's random input, which the receiver should not learn.

Now, we define unpredictability for π^{OT}:

In this definition, we capture the situation where a (semi-)honestly corrupted receiver has to guess the sender's input $y_{\bar{r}}$, which it did not receive. We assume that this input is random. We require that the receiver cannot guess $y_{\bar{r}}$ even if it receives the sender's leakage.

Definition 24. *Let* π^{OT} *be an* OT *protocol, and* $\mathsf{L}^{\mathsf{S}}_{\mathsf{OT}}(\cdot,\cdot)$ *be the leakage of its implementation. We say that the implementation of* π^{OT} *is* (q_G, t, ϵ)-unpredictable in the presence of leakage for a random input for a (semi)-honestly corrupted receiver, *denoted as* OT–OT–upL *if for any* t*-adversary* $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$,

$$\Pr[y_{\bar{r}} \in \mathcal{G} \mid |\mathcal{G}| \leq q_G,\ \mathcal{G} \leftarrow \mathsf{A}_2(\mathsf{st}, \mathsf{pp}^{\mathsf{OT}}, \mathsf{trans}_{\mathsf{OT}}, \mathsf{L}^{\mathsf{S}}_{\mathsf{OT}}(y_0, y_1, r), \mathsf{st}^{\mathsf{R}}_{fin}, R^{\mathsf{R}}),$$
$$(\mathsf{pp}^{\mathsf{OT}}, \mathsf{trans}_{\mathsf{OT}}, \mathsf{L}^{\mathsf{S}}_{\mathsf{OT}}(y_0, y_1, r) \leftarrow \pi^{\mathsf{OT}}\mathsf{L}(1^{\kappa}(y_0, y_1), r),$$
$$y_r \leftarrow y,\ y_{\bar{r}} \xleftarrow{\$} \{0,1\}^n,\ (\mathsf{st}, r, y) \leftarrow \mathsf{A}_1] \leq \epsilon,$$

where $\mathsf{st}^{\mathsf{R}}_{fin}$ *is the final state of the receiver (see Sec. 3.1), and* R^{R} *is the randomness used by the receiver to execute* π^{OT}.

We recall that we only need π^{OT} to be unpredictable, and we use it to build $\pi^{\mathsf{LR\text{-}OT}}$ which provides stronger indistinguishability security (Def. 16). We observe that guessing $y_{\bar{r}}$ is equivalent to guess $k_{\bar{r}}$ since $y_{\bar{r}} = z_{\bar{r}} \oplus k_{\bar{r}}$ (and $z_{\bar{r}} = \mathsf{F}_k(P_{\bar{r}})$ with $P_{\bar{r}}$ public and k known to the adversary). We therefore consider the following additional composition to be secure:

– \$, $\oplus$, F, and OT: \$ generates $k_{\bar{r}}$ (Step 2a), $\oplus$ computes $z_{\bar{r}} \oplus k_{\bar{r}}$ (Step 2b), F uses $k_{\bar{r}}$ as a key (Step 2c), π^{OT} uses $y_{\bar{r}}$ as input the receiver should not receive (Step 3).

Therefore, we need an additional security definition:

For \$, $\oplus$, F *and* π^{OT}. In our protocol, the sender picks $k_{\bar{r}}$, computes $y_{\bar{r}} = z_{\bar{r}} \oplus k_{\bar{r}}$, where $z_{\bar{r}}$ is chosen by the adversary in the security definition below, uses it as the key for F and then uses $y_{\bar{r}}$ as the sender's input for π^{OT} that the receiver should not receive. We require that $k_{\bar{r}}$ and $y_{\bar{r}}$ are unpredictable (note that learning one is equivalent to learning the other as $y_{\bar{r}} = k_{\bar{r}} \oplus z_{\bar{r}}$). As before, we are interested in the security of the composition of these four sub-computations since they are all related to $k_{\bar{r}}$ or $y_{\bar{r}}$. This compilation captures a secret that is being sent via an OT protocol. To generate this secret, we use two simple functions: random sampling and an XOR function, both of which are easy to protect against leakage. Similar to what was shown before, we can randomly sample shares of the key rather than using it directly to protect the implementation of F. Moreover, protecting the XOR function is trivial if the masking is additive [8]. For these reasons, we expect that the additional leakage of \$, $\oplus$, and F will not help guess the sender's input of π^{OT}. As before, we start from the q-upL definition, giving to the adversary the leakage of the random sampling, the leakage of the XOR, and the transcription and the leakage of the π^{OT} protocol. Similarly to Def. 22, we allow the adversary to choose the input known to the adversary, that is z, the value XORed to the key, and all the other inputs of the OT protocol, y_r and r. Formally,

Definition 25. *Let* π^{OT} *be an* OT *protocol,* \$ *be the random sampling,* $\oplus$ *be the XOR function, and* $\mathsf{L}^{\mathsf{S}}_{\mathsf{OT}}(\cdot,\cdot)$*,* $\mathsf{L}_{\$}(x)$*, and* $\mathsf{L}_{\oplus}(x,y)$ *be the leakage of their respective implementations. Let* $\mathsf{F} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ *be an ideal cipher and* L_{F} *be its leakage function. Let* \$ *generate* k*, which is the key used by the oracle* FL*. Let* k *be XORed to an adversarial chosen value* z*, obtaining* $y = (x \oplus k)$*. Let* y *be the* $y_{\bar{r}}$ *input of the* π^{OT} *protocol with* r *and* y_r *chosen by the adversary. We say that the implementation of the composition of the random sampling, the XOR,* π^{OT}*, and* F *is* $(q, q_G, q_{\mathsf{F}}, t, \epsilon)$-unpredictable in the presence of leakage $\mathsf{L}_{\$}(x), \mathsf{L}_{\oplus}(x,y), \mathsf{L}_{\mathsf{F}}(\cdot;\cdot), \mathsf{L}^{\mathsf{S}}_{\mathsf{OT}}(\cdot,\cdot)$ with π^{OT} using a random input*, denoted as* q-upL-$\mathsf{\$XORL}$-$\mathsf{OT}$*, if for any* (q, q_{F}, t)*-adversary* $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$*,*

$$\Pr[k \in \mathcal{G} \mid |\mathcal{G}| \leq q_G,\ \mathcal{G} \leftarrow \mathsf{A}_2^{\mathsf{FL}_k(\cdot),\mathsf{F}.(\cdot)}(\mathsf{st}, \mathsf{L}_{\$}(k), \mathsf{L}_{\oplus}(z,k), \mathsf{pp}^{\mathsf{OT}}, \mathsf{trans}_{\mathsf{OT}}, \mathsf{L}^{\mathsf{S}}_{\mathsf{OT}}(y_0, y_1, r)),$$
$$(\mathsf{pp}^{\mathsf{OT}}, \mathsf{trans}_{\mathsf{OT}}, \mathsf{L}^{\mathsf{S}}_{\mathsf{OT}}(y_0, y_1, r) \leftarrow \pi^{\mathsf{OT}}\mathsf{L}(1^{\kappa}(y_0, y_1), r),$$
$$y_r \leftarrow y,\ y_{\bar{r}} = z \oplus k,\ (k, \mathsf{L}_{\$}(k)) \xleftarrow{\$} \{0,1\}^n\ (\mathsf{st}, z, r, y) \leftarrow \mathsf{A}_1^{\mathsf{F}.(\cdot)}] \leq \epsilon,$$

with q_{F} *and the oracle* F *defined as in Def. 6.*

[8] It is enough to XOR z with one share and keep all other shares to have an additive output sharing.

In our case $q = 1$. Note that differently from the previous case (Def. 22) $z_{\bar{r}}$ is here known by the adversary (which knows k and $P_{\bar{r}}$), but not $y_{\bar{r}}$.

OT-S-LR-security of $\pi^{\mathsf{LR\text{-}OT}}$. Now, we can finally state and prove the OT-S-LR-security of $\pi^{\mathsf{LR\text{-}OT}}$:

Theorem 3. *Let* $\mathsf{F} : \{0,1\}^{n_{\mathsf{ENC}}} \times \{0,1\}^{n_{\mathsf{ENC}}} \to \{0,1\}^{n_{\mathsf{ENC}}}$ *be an ideal blockcipher. Let* F *be* $(1, q_G, q_{1,F}, t_1, \epsilon_{\text{1-upL-\$XORL-OT}})$ *-1-upL-secure with the key randomly sampled and XORed to a random value and* π^{OT} *sending the XOR. Let* ENC *be* $(q_{3,\mathsf{F}}, t_3, \epsilon_{\mathsf{EavL\text{-}GenFL}})$*-EavL-GenFL with the key is picked via* F*, then, for any* (q_{F}, t) *adversary* $\mathsf{A} = (\mathsf{A}_1^{\mathsf{F}}, \mathsf{A}_2^{\mathsf{F}})$*-adversary,* $\pi^{\mathsf{LR\text{-}OT}}$ *is* $(q_{\mathsf{F}}, t, \epsilon)$*-OT-S-LR secure, with*

$$\epsilon \leq \epsilon_{\text{1-upL-\$XORL-OT}} + \epsilon_{\mathsf{EavL\text{-}GenFL}}$$

where $t_1 = t + t_{\mathsf{OT}} + 2t_{\mathsf{Enc}}$*,* $t_2 = t + t_{\mathsf{OT}} + 2t_{\mathsf{Enc}}$*,* $t_3 = t + t_{\mathsf{OT}} + t_{\mathsf{Enc}}$*,* $q_{1,F} = 2 + q_{\mathsf{F}}$*,* $q_{2,F} = 1 + q_{\mathsf{F}}$*, and* $q_{3,F} = q_{\mathsf{F}}$*.*

Proof Idea: Our adversary has to distinguish whether the sender is sending (m_0^0, m_1^0) or (m_0^1, m_1^1), with $m_r^0 = m_r^1$ (Def. 16). Roughly speaking, if the adversary cannot guess $k_{\bar{r}}$ (1-upL-\$XORL-OT), the key $k_{\bar{r},E}$ remains random. Relying on the EavL-GenFL-security of ENC, an adversary cannot distinguish whether $c_{\bar{r}}$ is an encryption of $m_{\bar{r}}^0$ or $m_{\bar{r}}^1$.

Proof. The proof is inspired by those in the same model for F in [9].

Hybrids. We use a sequence of Hybrids, Hybrid 0, ..., Hybrid 3 where an adversary A is playing. Let E_i be the event that the adversary A outputs 1 at the end of Hybrid i.

– **Hybrid 0:** It is the OT-S-LR Hybrid where A is playing against $\pi^{\mathsf{LR\text{-}OT}}$, where the bit picked is 0.
– **Hybrid 1:** It is Hybrid 2, where we replace $y_{\bar{r}}, k_{\bar{r},E}$ with two random values and their leakage accordingly.
– **Hybrid 2:** It is Hybrid 1, where we replace $c_{\bar{r}}$ with $\mathsf{Enc}_{k_{\bar{r},E}}(m_{\bar{r}}^1)$ (that is, instead of encrypting $m_{\bar{r}}^0$, we encrypt $m_{\bar{r}}^1$), and its leakage accordingly.
– **Hybrid 3:** It is Hybrid 2, where we replace $y_{\bar{r}}$ and $k_{\bar{r},E}$ with the correct values, and the leakage accordingly.

Indistinguishability of Hybrids. Each of the previous Hybrids is indistinguishable from the following, as we now prove:

– **From Hybrid 0 to 1:** We observe the two Hybrids are the same except if the adversary A does a query to F with the key $k_{\bar{r}}$ because it would cause the loss of randomness of $k_{\bar{r},E}$. Thus,

$$|\Pr[E_0] - \Pr[E_1]| \leq \epsilon_{\text{1-upL-\$XORL-OT}}$$

– **From Hybrid 1 to 2:** Since ENC is $(q_{3,\mathsf{F}}, t_3, \epsilon_{\mathsf{EavL\text{-}GenFL}})$ and the result of Lemma 1

$$|\Pr[E_1] - \Pr[E_2]| \leq 2\epsilon_{\mathsf{EavL\text{-}GenFL}}$$

- **From Hybrid 2 to 3:** It is the transition between Hybrids 0 and 1 done in the opposite order. Thus,

$$|\Pr[E_2] - \Pr[E_3]| \leq \epsilon_{\mathsf{1\text{-}upL\text{-}\$XORL\text{-}OT}}$$

Now, we observe that Hybrid 0 is the $\mathsf{OT{-}S{-}LR}$ game where the bit picked is 0, while Hybrid 3 is the $\mathsf{OT{-}S{-}LR}$ game where the bit is picked 1. Thus,

$$|\Pr[E_0] - \Pr[E_3]| \leq 2\epsilon_{\mathsf{EavL\text{-}GenFL}} + 2\epsilon_{\mathsf{1\text{-}upL\text{-}\$XORL\text{-}OT}}.$$

Using Lemma 1

$$|\Pr[\mathsf{A} \text{ wins}] - 1/2| \leq 1/2(\Pr[E_0] - \Pr[E_3]) \leq \epsilon_{\mathsf{1\text{-}upL\text{-}\$XORL\text{-}OT}} + \epsilon_{\mathsf{EavL\text{-}GenFL}}.$$

□

5 A Leakage-Resilient Sequential OT

In this section, we discuss how to extend the previous construction in the case of multiple instances of OT. We work with a sequential composition because it allows us to use the previous instance's keys to encrypt the new instance's keys (thus avoiding using the key exchange protocol for all instances except for the first one). Note that in each iteration, the sender knows two keys $k_{i-1,0}, k_{i-1,1}$, while the receiver only knows $k_{i-1,r_{i-1}}$ (where r_{i-1} unknown to the sender). Thus, S needs to encrypt $k_{i,0}$ and $k_{i,1}$ under both keys $k_{i-1,0}, k_{i-1,1}$. Parallel execution would prevent this, as discussed in the next section.

For the first OT instance the $\pi^{\mathsf{LR\text{-}OT}}$ protocol is used as is, whereas for the ith instances, $i > 1$ the $\pi^{\mathsf{LR\text{-}OT}}$ protocol is used with the following differences:

- Step 1 is skipped (S and R do not execute π^{KE}).
- Step 2b is changed: S first refreshes the keys $k_{i-1,0}$ and $k_{i-1,1}$ by generating $k_{i-1,0,S}$, and $k_{i-1,1,S}$, where $k_{i-1,j,S} = \mathsf{F}_{k_i}(P_3)$, and P_3 a public constant. Furthermore, S encrypts both $k_{i,0}$ and $k_{i,1}$ keys with both $k_{i-1,0,S}$ and $k_{i-1,1,S}$ to get $y_{i,00}, y_{i,01}, y_{i,10}, y_{i,11}$, where $y_{i,jj'} = \mathsf{F}_{k_{i-1,j,S}}(P_{j'}) \oplus k_{i,j'}$.
- In Step 3, the sender's inputs for $\pi^{\mathsf{LR\text{-}OT}}$ are $((y_{i,00}, y_{i,10}), (y_{i,01}, y_{i,11}))$.
- In Step 4, the receiver computes $k_{\mathsf{R},i-1,S} = \mathsf{F}_{k_{\mathsf{R},i}}(P_3)$, and $k_{\mathsf{R},i} = y_{i,r_{i-1}r_i} \oplus \mathsf{F}_{k_{\mathsf{R},i-1,S}}(P_{r_i})$.

The full details of $\pi^{N\text{-}\mathsf{LR\text{-}OT}}$ are given in Alg. 2.

As per our modifications, we are using the same building blocks as for $\pi^{\mathsf{LR\text{-}OT}}$.

The protocol's correctness follows from the correctness of its underlying building blocks.

5.1 Black-Box Simulation-Based Security

Essentially, for simulation-based security, we only have to consider that each key $k_{i,j}$ is used twice as a key of F instead of once, and that there are more keys produced and used by F. We now give the formal result:

Theorem 4. *Let π^{KE} be a $(t_1, \epsilon_{\mathsf{KE}})$-secure key-exchange protocol (Def. 10), let* F *be a $(2, t_2, \epsilon_{\mathsf{PRF}})$-*PRF *(Def. 4), let* ENC *be a $(t_3, \epsilon_{\mathsf{Eav}})$-eavesdropper-secure encryption scheme (Def. 8), and let π^{OT} be a protocol that $(t_4, \epsilon_{\mathsf{OT}})$-securely computes*

Algorithm 2 Our N-OT protocol $\pi^{N\text{-}\mathsf{LR\text{-}OT}} = (\pi^{N\text{-}\mathsf{LR\text{-}OT}}.\mathsf{Setup}, \pi^{N\text{-}\mathsf{LR\text{-}OT}}.\mathsf{OT})$.

```
– Building blocks:
    1. $\pi^{\mathsf{KE}} = (\pi^{\mathsf{KE}}.\mathsf{Setup}, \pi^{\mathsf{KE}}.\mathsf{KE})$, a key-exchange protocol,
    2. $\mathsf{F}$, a block-cipher,
    3. $\mathsf{ENC} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, an $\mathsf{EavL}$-secure encryption scheme,
    4. $\pi^{\mathsf{OT}} = (\pi^{\mathsf{OT}}.\mathsf{Setup}, \pi^{\mathsf{OT}}.\mathsf{OT})$, an $\mathsf{OT}$ protocol.
– Input: $\mathsf{S}$ has $n$-couples of strings $M := ((m_{1,0}, m_{1,1}), ..., (m_{N,0}, m_{N,1}))$ with
  $|m_{i,0}| = |m_{i,1}|\ \forall i \in [N]$; $\mathsf{R}$ has a string $r = (r_1, ..., r_N) \in \{0,1\}^N$
– Auxiliary input: $1^\kappa$ the security parameter (shared by both $\mathsf{S}$ and $\mathsf{R}$)
– Setup phase: $\pi^{\mathsf{LR\text{-}OT}}.\mathsf{Setup}(1^\kappa)$ does:
    1. From $1^\kappa$ choose $n_{\mathsf{ENC}}$, a $\mathsf{BC}$ $\mathsf{F} : \{0,1\}^{n_{\mathsf{ENC}}} \times \{0,1\}^{n_{\mathsf{ENC}}} \to \{0,1\}^{n_{\mathsf{ENC}}}$, an $\mathsf{EavL}$-
       secure encryption scheme $\mathsf{ENC}$, four different strings $P_0, P_1, P_2, P_3 \in \{0,1\}^{n_{\mathsf{ENC}}}$.
    2. $\mathsf{pp}^{\mathsf{KE}} \leftarrow \pi^{\mathsf{KE}}.\mathsf{Setup}(1^\kappa, n_{\mathsf{ENC}})$,
    3. $\mathsf{pp}^{\mathsf{OT}} \leftarrow \pi^{\mathsf{OT}}.\mathsf{Setup}(1^\kappa)$
    4. Return $\mathsf{pp} = (\mathsf{pp}^{\mathsf{KE}}, \mathsf{pp}^{\mathsf{OT}}, n_{\mathsf{ENC}}, \mathsf{F}, \mathsf{ENC}, P_0, P_1, P_2)$
– $\pi^{\mathsf{LR\text{-}OT}}.\mathsf{OT}(\mathsf{pp})$:
– First $\mathsf{OT}$ instance:
    1. Key-exchange phase: $\mathsf{S}$ and $\mathsf{R}$ execute $\pi^{\mathsf{KE}}.\mathsf{KE}(\mathsf{pp}^{\mathsf{KE}})$ to both obtain $k$
    2. Sender's phase:
        (a) $\mathsf{S}$ picks $k_{1,0}, k_{1,1} \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$
        (b) $\mathsf{S}$ computes $z_{1,0} = \mathsf{F}_k(P_0)$, $y_{1,0} = z_{1,0} \oplus k_{1,0}$, $z_{1,1} = \mathsf{F}_k(P_1)$, $y_{1,1} =$
            $z_{1,1} \oplus k_{1,1}$,
        (c) $\mathsf{S}$ computes $k_{1,0,E} = \mathsf{F}_{k_{1,0}}(P_2)$, $k_{1,1,E} = \mathsf{F}_{k_{1,1}}(P_2)$,
        (d) $\mathsf{S}$ computes $c_{1,0} = \mathsf{Enc}_{k_{1,0,E}}(m_{1,0})$, $c_{1,1} = \mathsf{Enc}_{k_{1,1,E}}(m_{1,1})$
        (e) $\mathsf{S}$ sends $c_{1,0}, c_{1,1}$ to $\mathsf{R}$
    3. Send-key phase: $\mathsf{S}$ and $\mathsf{R}$ execute $\pi^{\mathsf{OT}}.\mathsf{OT}(\mathsf{pp}^{\mathsf{OT}})$ with $\mathsf{S}$'s input $(y_{1,0}, y_{1,1})$
       and $\mathsf{R}$'s $r_1$
    4. Receiver's phase:
        $\mathsf{R}$ computes $k_{\mathsf{R},1} = \mathsf{F}_k(P_r) \oplus y_{1,r_1}$, $k_{\mathsf{R},1,E} = \mathsf{F}_{k_{\mathsf{R},1}}(P_2)$,
        $\mathsf{R}$ computes $m_1 = \mathsf{Dec}_{k_{\mathsf{R},1,E}}(c_{1,r_1})$
– Remaining $\mathsf{OT}$ instances: For $i = 2, \ldots, N$
    1. Sender's phase:
        (a) $\mathsf{S}$ picks $k_{i,0}, k_{i,1} \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$
        (b) $\mathsf{S}$ computes $k_{i-1,0,S} = \mathsf{F}_{k_{i-1,0}}(P_3)$, $k_{i-1,1,S} = \mathsf{F}_{k_{i-1,1}}(P_3)$, computes
            $z_{i,00} = \mathsf{F}_{k_{i-1,0,S}}(P_0)$, $y_{i,00} = z_{i,00} \oplus k_{i,0}$, $z_{i,01} = \mathsf{F}_{k_{i-1,0,S}}(P_1)$,
            $y_{i,01} = z_{i,01} \oplus k_{i,1}$, $z_{i,10} = \mathsf{F}_{k_{i-1,1,S}}(P_0)$, $y_{i,10} = z_{i,10} \oplus k_{i,0}$,
            $z_{i,11} = \mathsf{F}_{k_{i-1,1,S}}(P_1)$, $y_{i,11} = z_{i,11} \oplus k_{i,11}$
        (c) $\mathsf{S}$ computes $k_{i,0,E} = \mathsf{F}_{k_{i,0}}(P_2)$, $k_{i,1,E} = \mathsf{F}_{k_{i,1}}(P_2)$,
        (d) $\mathsf{S}$ computes $c_{i,0} = \mathsf{Enc}_{k_{i,0,E}}(m_{i,0})$, $c_{i,1} = \mathsf{Enc}_{k_{i,1,E}}(m_{i,1})$
        (e) $\mathsf{S}$ sends $c_{i,0}, c_{i,1}$ to $\mathsf{R}$
    2. Send-key phase: $\mathsf{S}$ and $\mathsf{R}$ execute $\pi^{\mathsf{OT}}.\mathsf{OT}(\mathsf{pp}^{\mathsf{OT}})$ with $\mathsf{S}$'s input
       $((y_{i,00}, y_{i,10}), (y_{i,01}, y_{i,11}))$ and $\mathsf{R}$'s $r_i$
    3. Receiver's phase:
        $\mathsf{R}$ computes $k_{\mathsf{R},i-1,S} = \mathsf{F}_{k_{\mathsf{R},i-1}}(P_3)$, $k_{\mathsf{R},i} = \mathsf{F}_{k_{\mathsf{R},i-1,S}}(P_{r_i}) \oplus y_{i,r_{i-1}r_i}$, $k_{\mathsf{R},i,E} =$
            $\mathsf{F}_{k_{\mathsf{R},i}}(P_2)$,
        $\mathsf{R}$ computes $m_i = \mathsf{Dec}_{k_{\mathsf{R},i,E}}(c_{i,r_i})$
– Output: $\mathsf{S}$: nothing; $\mathsf{R}$: $M_{\mathsf{R}} = (m_1, ..., m_N)$
```

the OT *functionality (Def. 11,Sec. 2.6), then, the* $\pi^{N\text{-}\mathsf{LR}\text{-}\mathsf{OT}}$ *protocol defined in Alg. 2,* (t,ϵ)*-securely computes the* OT *functionality with*

$$\epsilon \leq \epsilon_{\mathsf{KE}} + N(\epsilon_{\mathsf{OT}} + 2\epsilon_{\mathsf{PRF}} + \epsilon_{\mathsf{Eav}}),$$

provided that $|m_0|, |m_1| \leq B$ *bits, where* $t_1 = t + N(t_{\mathsf{OT}} + 4t_{\mathsf{F}} + 2t_{\mathsf{Enc}})$*,* $t_2 = t + t_{\mathsf{Sim,KE}} + N(t_{\mathsf{Sim,OT}} + 2t_{\mathsf{F}} + 2t_{\mathsf{Enc}})$*,* $t_3 = t + t_{\mathsf{Sim,KE}} + N(t_{\mathsf{Sim,OT}} + 2t_{\mathsf{F}} + t_{\mathsf{Enc}})$*,* $t_4 = t + t_{\mathsf{Sim,KE}} + N(t_{\mathsf{OT}} + 4t_{\mathsf{F}} + 2t_{\mathsf{Enc}})$*,* $t_{\mathsf{Sim,KE}}$ *is the time needed to simulate the* π^{KE} *protocol,* t_{OT} *to execute the* π^{OT} *protocol,* $t_{\mathsf{Sim,OT}}$ *to simulate the* π^{OT} *protocol,* t_{F} *to execute* F*, and* t_{Enc} *to encrypt with* Enc *a message of at most* B *bits.*

Proof. We start by building the simulator for the sender, $\mathsf{Sim}_{\mathsf{S}}$ and by proving that the view given by $\mathsf{Sim}_{\mathsf{S}}$ is (t,ϵ)-computationally indistinguishable from the real view. Then, we build $\mathsf{Sim}_{\mathsf{R}}$, and we prove that it is a (t,ϵ)-simulator.

Notations. Let $M = ((m_{1,0}, m_{1,1}), \dots, (m_{N,0}, m_{N,1})$ be the input of the sender and $r = (r_1, \dots, r_n)$ be the input of the receiver. Let $M_r = (m_{1,r_1}, \dots, m_{N,r_N})$ be the receiver's output.

Since π^{KE} is $(t_1, \epsilon_{\mathsf{KE}})$-secure, there are two simulators $\mathsf{Sim}_{\mathsf{S}}^{\mathsf{KE}}$, $\mathsf{Sim}_{\mathsf{R}}^{\mathsf{KE}}$ s.t. the joint distribution of the $(\mathsf{view}_{\mathsf{S}}^{\pi^{\mathsf{KE}}}(1^\kappa, n_{\mathsf{ENC}}), k')$ [resp. $(\mathsf{view}_{\mathsf{R}}^{\pi^{\mathsf{KE}}}(1^\kappa, n_{\mathsf{ENC}}), k')$], where k' is the output of the π^{KE}-protocol, is ϵ_{KE}-computationally indistinguishable from $(\mathsf{Sim}_{\mathsf{S}}^{\mathsf{KE}}(1^\kappa, n_{\mathsf{ENC}}, k), k)$ [resp. $(\mathsf{Sim}_{\mathsf{R}}^{\mathsf{KE}}(1^\kappa, n_{\mathsf{Enc}}, k), k)$] with $k \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$. Similarly, since π^{OT} is $(t_4, \epsilon_{\mathsf{OT}})$-secure, there are two simulators $\mathsf{Sim}_{\mathsf{S}}^{\mathsf{OT}}$, $\mathsf{Sim}_{\mathsf{R}}^{\mathsf{OT}}$ s.t. $\mathsf{view}_{\mathsf{S}}^{\pi^{\mathsf{OT}}}(1^\kappa, (m_0, m_1), \perp)$ [resp. $(\mathsf{view}_{\mathsf{R}}^{\pi^{\mathsf{OT}}}(1^\kappa, r, m_r)$] is ϵ_{OT}-computationally indistinguishable from $(\mathsf{Sim}_{\mathsf{S}}^{\mathsf{OT}}(1^\kappa, (x_0, x_1), \perp)$ [resp. $\mathsf{Sim}_{\mathsf{R}}^{\mathsf{OT}}(1^\kappa, r, x_r)$].

Sender (semi-honestly) corrupted.
We simulate the sender's view with $\mathsf{Sim}_{\mathsf{S}}(1^\kappa, M, \perp)$. It works like this:

- **Setup-phase:** $\mathsf{Sim}_{\mathsf{S}}(1^\kappa)$ chooses n_{ENC}, F, ENC, and four different strings in $\{0,1\}^{n_{\mathsf{ENC}}}$: P_0, P_1, P_2, and P_3. Then it runs $\mathsf{Sim}_{\mathsf{S}}^{\mathsf{KE.Setup}}(1^\kappa, n_{\mathsf{ENC}})$ obtaining $(\mathsf{pp}^{\mathsf{KE}}, \mathsf{view}_{\mathsf{S}}^{\pi^{\mathsf{KE}}.\mathsf{Setup}})$, and $\mathsf{Sim}_{\mathsf{S}}^{\mathsf{OT.Setup}}(1^\kappa)$ obtaining $(\mathsf{pp}^{\mathsf{OT}}, \mathsf{view}_{\mathsf{S}}^{\pi^{\mathsf{OT}}.\mathsf{Setup}})$, respectively. $\mathsf{Sim}_{\mathsf{S}}$ defines
$$\mathsf{view}_{\mathsf{S}}^{\mathsf{Setup}} := ((\mathsf{pp}^{\mathsf{KE}}, \mathsf{pp}^{\mathsf{OT}}, n_{\mathsf{ENC}}, \mathsf{F}), \mathsf{view}_{\mathsf{S}}^{\pi^{\mathsf{KE}}.\mathsf{Setup}}, \mathsf{view}_{\mathsf{S}}^{\pi^{\mathsf{OT}}.\mathsf{Setup}}).$$
- **First** OT **instance**:
  - **Key-exchange phase:** $\mathsf{Sim}_{\mathsf{S}}$ picks $k \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$ and runs $\mathsf{Sim}_{\mathsf{S}}^{\mathsf{KE}}(1^\kappa, \mathsf{pp}^{\mathsf{KE}}, k)$ obtaining $\mathsf{view}_{\mathsf{S}}^{\mathsf{KE.KE}}$.
  - **Sender's phase:** Using S'random tape, $\mathsf{Sim}_{\mathsf{S}}$ can pick correctly $k_{1,0}, k_{1,1} \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$. Then, it computes $(y_{1,0}, y_{1,1})$ as in Alg 2, $y_{1,i} = \mathsf{F}_k(P_i) \oplus k_{1,i}$.
  - **Send key-phase:** $\mathsf{Sim}_{\mathsf{S}}$ computes
    $\mathsf{view}_{\mathsf{S}}^{\mathsf{OT},1} := \mathsf{Sim}_{\mathsf{S}}^{\mathsf{OT.OT}}(1^\kappa, \mathsf{pp}^{\mathsf{OT}}, (y_{1,0}, y_{1,1}), \perp)$.
- **Remaining** OT **instance:** For $j = 2, \dots, N$
  - **Sender's phase:** Using S'random tape, $\mathsf{Sim}_{\mathsf{S}}$ can pick correctly $k_{j,0}, k_{j,1} \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$. Then, it computes $k_{j-1,i,S} = \mathsf{F}_{k_{j-1},i}(P_3)$, $(y_{j,00}y_{j,10}, y_{j,01}, y_{j,11})$ as in Alg 2, $y_{j,i'i} = \mathsf{F}_{k_{j-1,i',S}}(P_i) \oplus k_{j,i}$.
  - **Send key-phase:** $\mathsf{Sim}_{\mathsf{S}}$ computes
    $\mathsf{view}_{\mathsf{S}}^{\mathsf{OT},j} := \mathsf{Sim}_{\mathsf{S}}^{\mathsf{OT.OT}}(1^\kappa, \mathsf{pp}^{\mathsf{OT}}, ((y_{j,00}, y_{j,10}), (y_{j,01}, y_{j,11})), \perp)$.

– **Simulator output:** At the end, Sim_S outputs $\mathsf{view}_\mathsf{S}^{\mathsf{Sim}}$ where
$$\mathsf{view}_\mathsf{S}^{\mathsf{Sim}} := (\mathsf{view}_\mathsf{S}^{\mathsf{Setup}}, \mathsf{view}_\mathsf{S}^{\mathsf{KE.KE}}, \mathsf{view}_\mathsf{S}^{\mathsf{OT},1}, \ldots, \mathsf{view}_\mathsf{S}^{\mathsf{OT},N}).$$

Hybrids. To prove that the simulator Sim_S outputs a view that is indistinguishable from the real execution of the protocol, we introduce three different view distributions.

– Let $H_\mathsf{S}^0(1^\kappa, M, r)$ be S's view distribution of the real execution of $\pi^{\mathsf{LR\text{-}OT}}$.
– We modify the previous view distribution obtaining the view distribution $H_\mathsf{S}^1(1^\kappa, M, r)$ by simulating the execution of the key-exchange protocol, using $\mathsf{Sim}_\mathsf{S}^{\mathsf{KE}}$, instead of having the real execution. We use $\mathsf{Sim}_\mathsf{S}^{\mathsf{KE}}(1^\kappa, n_{\mathsf{ENC}}, \mathsf{pp}^{\mathsf{KE}}, k)$, where 1^κ is the security parameter, , n_{ENC} obtained as in the real execution of the protocol and $k \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$ (π^{KE} is a key-exchange protocol (Def. 12), that is, it implements the functionality $k \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$).
– We modify the previous view distribution obtaining the view distribution $H_\mathsf{S}^2(1^\kappa, M, r)$ by simulating the execution of the OT protocol in the first instance, using $\mathsf{Sim}_\mathsf{S}^{\mathsf{OT}}(1^\kappa, \mathsf{pp}^{\mathsf{OT}}, (y_{1,0}, y_{1,1}), \perp)$, instead of having the real execution. Sim_S can compute $y_{1,0}$ and $y_{1,1}$ correctly: accessing the random tape of S, Sim_S can pick $k_{1,0}, k_{1,1} \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$ correctly and compute $y_{1,i} = \mathsf{F}_k(P_i) \oplus k_{1,i}$.
– We have a sequence of view distributions $H_\mathsf{S}^3(1^\kappa, M, r), \ldots H_\mathsf{S}^{N+1}(1^\kappa, M, r)$. We modify the previous view distribution, $H_\mathsf{S}^j(1^\kappa, M, r)$, obtaining the view distribution $H_\mathsf{S}^{j+1}(1^\kappa, M, r)$ by simulating the execution of the OT protocol in the j^{th} instance, using $\mathsf{Sim}_\mathsf{S}^{\mathsf{OT}}(1^\kappa, \mathsf{pp}^{\mathsf{OT}}, ((y_{j,00}, y_{j,10}), (y_{j,01}, y_{j,11})), \perp)$, instead of having the real execution. Sim_S can compute $y_{j,i'i}$ correctly, for $i, i' \in \{0,1\}$: accessing the random tape of S, Sim_S can pick $k_{j,0}, k_{j,1} \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$ correctly and compute $y_{j,i'i} = \mathsf{F}_{k_{j-1,i',S}}(P_i) \oplus k_{j,i}$, with $k_{j-1,i',S} = \mathsf{F}_{j-1,i'}(P_3)$.

Indistinguishability of the hybrids. Now, we show that each view distribution is computationally indistinguishable from the next.

Since π^{KE} is a $(t_1, \epsilon_{\mathsf{KE}})$-secure key-exchange protocol, then
$$|\Pr[1 \leftarrow \mathsf{A}(H_\mathsf{S}^0(1^\kappa, M, r))] - \Pr[1 \leftarrow \mathsf{A}(H_\mathsf{S}^1(1^\kappa, M, r))]| \leq \epsilon_{\mathsf{KE}}.$$
Since π^{OT} is a $(t_4, \epsilon_{\mathsf{OT}})$-secure OT protocol, and $y_{1,0}$ and $y_{1,1}$ are computed as in the correct execution of $\pi^{\mathsf{LR\text{-}OT}}$, then
$$|\Pr[1 \leftarrow \mathsf{A}(H_\mathsf{S}^1(1^\kappa, M, r))] - \Pr[1 \leftarrow \mathsf{A}(H_\mathsf{S}^2(1^\kappa, M, r))]| \leq \epsilon_{\mathsf{OT}}.$$
Since π^{OT} is a $(t_4, \epsilon_{\mathsf{OT}})$-secure OT protocol, and $y_{j,00}, y_{j,10}, y_{j,01}$ and $y_{j,11}$ are computed as in the correct execution of $\pi^{\mathsf{LR\text{-}OT}}$, then
$$|\Pr[1 \leftarrow \mathsf{A}(H_\mathsf{S}^j(1^\kappa, M, r))] - \Pr[1 \leftarrow \mathsf{A}(H_\mathsf{S}^{j+1}(1^\kappa, M, r))]| \leq \epsilon_{\mathsf{OT}}.$$
This concludes the proof since $H_\mathsf{S}^{N+1}(1^\kappa, M, r)$ is the distribution of the views simulated by Sim_S. Thus, the real view distribution, $H_\mathsf{S}^0(1^\kappa, M, r)$, and the simulated one, $H_\mathsf{S}^{N+1}(1^\kappa, M, r)$ are (t, ϵ')-computationally indistinguishable, since
$$|\Pr[1 \leftarrow \mathsf{A}(H_\mathsf{S}^0(1^\kappa, M, r))] - \Pr[1 \leftarrow \mathsf{A}(H_\mathsf{S}^{N+1}(1^\kappa, M, r))]| \leq \epsilon_{\mathsf{KE}} + N\epsilon_{\mathsf{OT}} = \epsilon' \leq \epsilon.$$

Receiver (semi-honestly) corrupted.

We simulate the receiver's view with $\mathsf{Sim}_\mathsf{R}(1^\kappa, r, M_r)$. It works like this:

- **Setup-phase:** $\mathsf{Sim_R}(1^\kappa)$ chooses n_{ENC}, F, ENC, and four different strings in $\{0,1\}^{n_{\mathsf{ENC}}}$: $P_0,\ P_1,\ P_2$, and P_3. Then it runs $\mathsf{Sim}_\mathsf{R}^{\mathsf{KE.Setup}}(1^\kappa, n_{\mathsf{ENC}})$ obtaining $(\mathsf{pp}^{\mathsf{KE}}, \mathsf{view}_\mathsf{R}^{\pi^{\mathsf{KE}}.\mathsf{Setup}})$, and $\mathsf{Sim}_\mathsf{R}^{\mathsf{OT.Setup}}(1^\kappa)$ obtaining $(\mathsf{pp}^{\mathsf{OT}}, \mathsf{view}_\mathsf{R}^{\pi^{\mathsf{OT}}.\mathsf{Setup}})$, respectively. $\mathsf{Sim_R}$ defines
$$\mathsf{view}_\mathsf{R}^{\mathsf{Setup}} := ((\mathsf{pp}^{\mathsf{KE}}, \mathsf{pp}^{\mathsf{OT}}, n_{\mathsf{ENC}}, \mathsf{F}), \mathsf{view}_\mathsf{R}^{\pi^{\mathsf{KE}}.\mathsf{Setup}}, \mathsf{view}_\mathsf{R}^{\pi^{\mathsf{OT}}.\mathsf{Setup}}).$$
- **First OT instance**:
  - **Key-exchange phase:** $\mathsf{Sim_R}$ picks $k \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$ and runs $\mathsf{Sim}_\mathsf{R}^{\mathsf{KE}}(1^\kappa, \mathsf{pp}^{\mathsf{KE}}, k)$ obtaining $\mathsf{view}_\mathsf{R}^{\mathsf{KE.KE}}$.
  - **Sender's phase:** $\mathsf{Sim_R}$ picks $k_{1,r_1}, k_{1,\bar{r_1},E} \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$, and an arbitrary message $\tilde{m}_{1,\bar{r_1}}$ (for example, $\tilde{m}_{1,\bar{r_1}} \xleftarrow{\$} \{0,1\}^{|m_{1,r_1}|}$). It computes $y_{1,r_1} = \mathsf{F}_k(P_{r_1}) \oplus k_{1,r_1}$, $k_{1,r_1,E} = \mathsf{F}_{k_{1,r_1}}(P_2), c_{1,r_1} = \mathsf{Enc}_{k_{1,r_1,E}}(m_{1,r_1})$, $c_{1,\bar{r}_1} = \mathsf{Enc}_{k_{1,\bar{r_1},E}}(\tilde{m}_{1,\bar{r}_1})$.
  - **Send key-phase:** $\mathsf{Sim_R}$ computes $\mathsf{view}_\mathsf{R}^{\mathsf{OT},1} := \mathsf{Sim}_\mathsf{R}^{\mathsf{OT.OT}}(1^\kappa, \mathsf{pp}^{\mathsf{OT}}, r_1, y_{1,r_1})$.
- **Remaining OT instance:** For $j = 2, \ldots, N$
  - **Sender's phase:** $\mathsf{Sim_R}$ picks $k_{j,r_j}, k_{j,\bar{r_j},E}, y_{j,\bar{r}_{j-1}r_j} \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$, and an arbitrary message $\tilde{m}_{j,\bar{r_j}}$ (for example, $\tilde{m}_{j,\bar{r_j}} \xleftarrow{\$} \{0,1\}^{|m_{j,r_j}|}$). It computes $y_{j,r_{j-1}r_j} = \mathsf{F}_{k_{j-1,r_{j-1},S}}(P_{r_j}) \oplus k_{j,r_j}$, with $k_{j-1,r_{j-1},S} = \mathsf{F}_{k_{j-1,r_{j-1}}}(P_3)$, $k_{j,r_j,E} = \mathsf{F}_{k_{j,r_j}}(P_2), c_{j,r_j} = \mathsf{Enc}_{k_{j,r_j,E}}(m_{j,r_j})$, $c_{j,\bar{r}_j} = \mathsf{Enc}_{k_{j,\bar{r_j},E}}(\tilde{m}_{j,\bar{r}_j})$.
  - **Send key-phase:** $\mathsf{Sim_R}$ computes $\mathsf{view}_\mathsf{R}^{\mathsf{OT},j} := \mathsf{Sim}_\mathsf{R}^{\mathsf{OT.OT}}(1^\kappa, \mathsf{pp}^{\mathsf{OT}}, r_j, (y_{j,0r_j}, y_{j,1r_j})$.
- **Simulator output:** At the end, $\mathsf{Sim_R}$ outputs
$$\mathsf{view}_\mathsf{R}^{\mathsf{Sim}} := (\mathsf{view}_\mathsf{R}^{\mathsf{Setup}}, \mathsf{view}_\mathsf{R}^{\mathsf{KE.KE}}, c_{1,0}, c_{1,1}, \mathsf{view}_\mathsf{R}^{\mathsf{OT},1}, \ldots, c_{N,0}, c_{N,1}, \mathsf{view}_\mathsf{R}^{\mathsf{OT},N}).$$

Hybrids. To prove that the simulator $\mathsf{Sim_R}$ outputs a view that is indistinguishable from the real execution of the protocol, we introduce five different view distributions.

- Let $H_\mathsf{R}^0(1^\kappa, M, r)$ be R's view distribution of the real execution of $\pi^{\mathsf{LR\text{-}OT}}$.
- We modify the previous view distribution obtaining the view distribution $H_\mathsf{R}^{0,1}(1^\kappa, M, r)$ by simulating the execution of the key-exchange protocol, using $\mathsf{Sim}_\mathsf{R}^{\mathsf{KE}}$, instead of having the real execution. We use $\mathsf{Sim}_\mathsf{R}^{\mathsf{KE}}(1^\kappa, n_{\mathsf{ENC}}, k)$, where 1^κ is the security parameter, $, n_{\mathsf{ENC}}$ obtained as in the real execution of the protocol and $k \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$ (π^{KE} is a key-exchange protocol (Def. 12), that is, it implements the functionality $k \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$).
- We modify the previous view distribution obtaining the view distribution $H_\mathsf{R}^{0,2}(1^\kappa, M, r)$ by simulating the execution of the OT protocol, using $\mathsf{Sim}_\mathsf{R}^{\mathsf{OT}}$, instead of having the real execution. We use $\mathsf{Sim}_\mathsf{R}^{\mathsf{OT}}(1^\kappa, r_1, y_{1,r_1})$, where 1^κ is the security parameter, $y_{1,r_1} = \mathsf{F}_k(P_{r_1}) \oplus k_{1,r_1}$, with $k_{1,r_1} \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$.
- We modify the previous view distribution obtaining the view distribution $H_\mathsf{R}^{0,3}(1^\kappa, M, r)$ by picking $k_{1,\bar{r}_1,E}, k_{1,\bar{r}_1,S} \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$, instead of computing $k_{1,\bar{r}_1,E} = \mathsf{F}_{k_{1,\bar{r}_1}}(P_2)$, and $k_{1,\bar{r}_1,S} = \mathsf{F}_{k_{1,\bar{r}_1}}(P_3)$.

- We modify the previous view distribution obtaining the view distribution $H_{\mathsf{R}}^{0,4}(1^\kappa, M, r) := H_{\mathsf{R}}^{1}(1^\kappa, M, r)$ by computing $c_{1,\bar{r}_1} = \mathsf{Enc}_{k_{1,\bar{r}_1,E}}(\tilde{m}_{1,\bar{r}_1})$, where $\tilde{m}_{1,\bar{r}_1}$ is an arbitrary message (for example, $\tilde{m}_{1,\bar{r}_1} \xleftarrow{\$} \{0,1\}^{|m_{1,r_1}|}$).
- Then, we have a sequence of view distributions $H_{\mathsf{R}}^{1,1}(1^\kappa, M, r), \ldots, H_{\mathsf{R}}^{1,4}(1^\kappa, M, r), \ldots H_{\mathsf{R}}^{N-1,1}(1^\kappa, M, r), \ldots, H_{\mathsf{R}}^{N-1,4}(1^\kappa, M, r)$ with $H_{\mathsf{R}}^{j}(1^\kappa, M, r) := H_{\mathsf{R}}^{j,4}(1^\kappa, M, r)$.
  - We modify the previous view distribution, $H_{\mathsf{R}}^{j-1}(1^\kappa, M, r)$, obtaining the view distribution $H_{\mathsf{R}}^{j,1}(1^\kappa, M, r)$ by picking $y_{j,\bar{r}_{j-1}r_j} \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$ instead of computing it as $\mathsf{F}_{k_{j-1,\bar{r}_j},S}(P_{r_j}) \oplus k_{j,r_j}$, with $k_{j,r_j} \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$.
  - We modify the previous view distribution obtaining the view distribution $H_{\mathsf{R}}^{j-1,2}(1^\kappa, M, r)$ by simulating the execution of the OT protocol, using $\mathsf{Sim}_{\mathsf{R}}^{\mathsf{OT}}$, instead of having the real execution. We use $\mathsf{Sim}_{\mathsf{R}}^{\mathsf{OT}}(1^\kappa, r_j, (y_{j,0r_j}, y_{j,1r_j}))$, where 1^κ is the security parameter, $y_{j,r_{j-1}r_j} = \mathsf{F}_{k_{j-1,r_{j-1},S}}(P_{r_j}) \oplus k_{j,r_j}$, with $k_{j,r_j} \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$.
  - We modify the previous view distribution obtaining the view distribution $H_{\mathsf{R}}^{j-1,3}(1^\kappa, M, r)$ by picking $k_{j,\bar{r}_j,E}, k_{j,\bar{r}_j,S} \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$, instead of computing $k_{j,\bar{r}_j,E} = \mathsf{F}_{k_{j,\bar{r}_j}}(P_2)$, and $k_{j,\bar{r}_j,S} = \mathsf{F}_{k_{j,\bar{r}_j}}(P_3)$.
  - We modify the previous view distribution obtaining the view distribution $H_{\mathsf{R}}^{j-1,4}(1^\kappa, M, r) := H_{\mathsf{R}}^{j}(1^\kappa, M, r)$ by computing $c_{j,\bar{r}_j} = \mathsf{Enc}_{k_{j,\bar{r}_j,E}}(\tilde{m}_{j,\bar{r}_j})$, where $\tilde{m}_{j,\bar{r}_j}$ is an arbitrary message (for example, $\tilde{m}_{j,\bar{r}_j} \xleftarrow{\$} \{0,1\}^{|m_{j,r_j}|}$).

Indistinguishability of the hybrids. Now, we show that each view distribution is computationally indistinguishable from the next.

Since π^{KE} is a $(t_1, \epsilon_{\mathsf{KE}})$-secure key-exchange protocol, then

$$|\Pr[1 \leftarrow \mathsf{A}(H_{\mathsf{R}}^{0}(1^\kappa, M, r))] - \Pr[1 \leftarrow \mathsf{A}(H_{\mathsf{R}}^{0,1}(1^\kappa, M, r))]| \leq \epsilon_{\mathsf{KE}}.$$

Since π^{OT} is a $(t_4, \epsilon_{\mathsf{OT}})$-secure OT protocol, $\mathsf{Sim}_{\mathsf{R}}$ picks k_{1,r_1} with the same distribution as in the original protocol and computes y_{1,r_1} as in the correct execution of $\pi^{\mathsf{LR\text{-}OT}}$, then

$$|\Pr[1 \leftarrow \mathsf{A}(H_{\mathsf{R}}^{0,1}(1^\kappa, M, r))] - \Pr[1 \leftarrow \mathsf{A}(H_{\mathsf{R}}^{0,2}(1^\kappa, M, r))]| \leq \epsilon_{\mathsf{OT}}.$$

Since F is a $(2, t_2, \epsilon_{\mathsf{PRF}})$-$\mathsf{PRF}$ and $k_{1,\bar{r}_1}$ is picked randomly and it is secret for the receiver, then

$$|\Pr[1 \leftarrow \mathsf{A}(H_{\mathsf{R}}^{0,2}(1^\kappa, M, r))] - \Pr[1 \leftarrow \mathsf{A}(H_{\mathsf{R}}^{0,3}(1^\kappa, M, r))]| \leq \epsilon_{\mathsf{PRF}}.$$

Since ENC is a $(t_3, \epsilon_{\mathsf{Eav}})$-$\mathsf{Eav}$-secure encryption scheme and $k_{1,\bar{r}_1,E}$ is picked randomly and it is secret for the receiver, then

$$|\Pr[1 \leftarrow \mathsf{A}(H_{\mathsf{R}}^{0,3}(1^\kappa, M, r))] - \Pr[1 \leftarrow \mathsf{A}(H_{\mathsf{R}}^{0,4}(1^\kappa, M, r))]| \leq \epsilon_{\mathsf{Eav}}.$$

Now, for all the hybrids $H_{\mathsf{R}}^{1,1}(1^\kappa, M, r), \ldots, H_{\mathsf{R}}^{1,4}(1^\kappa, M, r), \ldots H_{\mathsf{R}}^{N,1}(1^\kappa, M, r), \ldots, H_{\mathsf{R}}^{N,4}(1^\kappa, M, r)$ with $H_{\mathsf{R}}^{j}(1^\kappa, M, r) := H_{\mathsf{R}}^{j-1,4}(1^\kappa, M, r)$, we can do the transitions as follows:

Since F is a $(2, t_2, \epsilon_{\mathsf{PRF}})$-$\mathsf{PRF}$ and $y_{j,\bar{r}_{j-1}r_j} = z_{j,\bar{r}_{j-1}r_j} \oplus k_{j,r_j}$ with $z_{j,\bar{r}_{j-1}r_j} = \mathsf{F}_{k_{j-1,\bar{r}_{j-1},S}}(P_{r_j})$ and $k_{j-1,\bar{r}_{j-1},S}$ is randomly picked (see the change between the view distribution $H_{\mathsf{R}}^{j-2,2}(1^\kappa, M, r)$ and the view distribution $H_{\mathsf{R}}^{j-2,3}(1^\kappa, M, r)$ is picked randomly and it is secret for the receiver, then

$$|\Pr[1 \leftarrow \mathsf{A}(H_{\mathsf{R}}^{j-1}(1^\kappa, M, r))] - \Pr[1 \leftarrow \mathsf{A}(H_{\mathsf{R}}^{j-1,1}(1^\kappa, M, r))]| \leq \epsilon_{\mathsf{PRF}}.$$

Since π^{OT} is a $(t_4, \epsilon_{\mathsf{OT}})$-secure OT protocol, $\mathsf{Sim}_{\mathsf{R}}$ picks k_{j,r_j} with the same distribution as in the original protocol and computes $y_{j,r_{j-1}r_j}$ as in the correct execution of $\pi^{\mathsf{LR\text{-}OT}}$, and $y_{j,\bar{r}_{j-1}r_j}$ is indistinguishable from the real one, then

$$|\Pr[1 \leftarrow \mathsf{A}(H_{\mathsf{R}}^{j-1,1}(1^\kappa, M, r))] - \Pr[1 \leftarrow \mathsf{A}(H_{\mathsf{R}}^{j-1,2}(1^\kappa, M, r))]| \leq \epsilon_{\mathsf{OT}}.$$

Since F is a $(2, t_2, \epsilon_{\mathsf{PRF}})$-$\mathsf{PRF}$ and $k_{j,\bar{r}_j}$ is picked randomly and it is secret for the receiver, then

$$|\Pr[1 \leftarrow \mathsf{A}(H_{\mathsf{R}}^{j-1,2}(1^\kappa, M, r))] - \Pr[1 \leftarrow \mathsf{A}(H_{\mathsf{R}}^{j-1,3}(1^\kappa, M, r))]| \leq \epsilon_{\mathsf{PRF}}.$$

Since ENC is a $(t_3, \epsilon_{\mathsf{Eav}})$-$\mathsf{Eav}$-secure encryption scheme and $k_{j,\bar{r}_j,E}$ is picked randomly and it is secret for the receiver, then

$$|\Pr[1 \leftarrow \mathsf{A}(H_{\mathsf{R}}^{j-1,3}(1^\kappa, M, r))] - \Pr[1 \leftarrow \mathsf{A}(H_{\mathsf{R}}^{j-1,4}(1^\kappa, M, r))]| \leq \epsilon_{\mathsf{Eav}}.$$

This concludes the proof since $H_{\mathsf{R}}^{N}(1^\kappa, M, r)$ is the distribution of the views simulated by $\mathsf{Sim}_{\mathsf{R}}$. Thus, the real view distribution, $H_{\mathsf{R}}^{0}(1^\kappa, M, r)$, and the simulated one, $H_{\mathsf{R}}^{N}(1^\kappa, M, r)$ are (t, ϵ)-computationally indistinguishable, since

$$|\Pr[1 \leftarrow \mathsf{A}(H_{\mathsf{R}}^{0}(1^\kappa, M, r))] - \Pr[1 \leftarrow \mathsf{A}(H_{\mathsf{R}}^{N}(1^\kappa, M, r))]| \leq$$
$$\epsilon_{\mathsf{KE}} + N\epsilon_{\mathsf{OT}} + (2N-1)\epsilon_{\mathsf{PRF}} + N\epsilon_{\mathsf{Eav}} \leq \epsilon_{\mathsf{KE}} + N(\epsilon_{\mathsf{OT}} + 2\epsilon_{\mathsf{PRF}} + \epsilon_{\mathsf{Eav}}) = \epsilon.$$

5.2 OT-S-LA -Security of $\pi^{N\text{-}\mathsf{LR\text{-}OT}}$

The proof against intercepting adversaries is slightly modified. First, we use two extra keys $k_{i-1,j,S}$ where these keys are obtained as an output of F, $k_{i-1,j,S} = \mathsf{F}_{k_{i_1,k}}(P_3)$, and are only used twice as the keys for F, $z_{i,jj'} = \mathsf{F}_{k_{i-1,j,S}}(P_{j'})$ (Step 2b). This re-keying scenario has been extensively treated in the literature [59, 45, 9]. Also, in Step 2b, $k_{i,j}$ is the input of two different XOR functions instead of one (for $i > 2$). Being the XOR the easiest function to protect against leakage, we believe that this is not a problem.

Now, we will give the details:

First, we have to adapt the composition definition for F, \$, and $\oplus$ function (Def. 22) to the case where 2 $\oplus$ functions are computed:

Definition 26. *Let \$ be the random sampling, $\oplus$ be the XOR function, and $\mathsf{L}_{\$}(x)$ and $\mathsf{L}_{\oplus}(x, y)$ be the leakage of their respective implementations. Let $\mathsf{F} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be an ideal cipher and L_{F} a leakage function. Let \$ generate k, used by the oracle FL. Let k be XORed to a random value z obtained by F. We say that the implementation of the composition of the random sampling, two XOR functions and F is $(q, q_G, q_{\mathsf{F}}, t, \epsilon)$-*unpredictable with leakage *$(\mathsf{L}_{\$}, \mathsf{L}_{\oplus}, \mathsf{L}_{\mathsf{F}})$, denoted as q-$\mathsf{upL}$-\$$\mathsf{XOR2L}$, if for any (q, q_{F}, t)-adversary* A,

$$\Pr[k \in \mathcal{G} \mid |\mathcal{G}| \leq q_G,$$
$$\mathcal{G} \leftarrow \mathsf{A}_2^{\mathsf{FL}_k(\cdot), \mathsf{F}.(\cdot)}(\mathsf{st}, \mathsf{L}_{\$}(k), y_0, \mathsf{L}_{\oplus}(z_0, k), \mathsf{L}_{\mathsf{F}}^{out}(k'_0; z_0), y_1, \mathsf{L}_{\oplus}(z_1, k), \mathsf{L}_{\mathsf{F}}^{out}(k'_1; z_1)),$$
$$y_0 = z_0 \oplus k,\ y_1 = z_1 \oplus k,\ (k, \mathsf{L}_{\$}(k)) \xleftarrow{\$} \{0,1\}^n, z_0, z_1 \xleftarrow{\$} \{0,1\}^n,$$
$$(\mathsf{st}, k'_0, k'_1) \leftarrow \mathsf{A}_1^{\mathsf{F}.(\cdot)}] \leq \epsilon,$$

with q_{F} and the oracle FL defined as in Def. 6.

Looking ahead, we will use this security definition twice: once to require that k_0 is not guessed and once for k_1. This formalization is coherent with the $\pi^{\mathsf{LR\text{-}OT}}$ flow. In our case, $q = 1$ or 2.

Theorem 5. *Let* $\mathsf{F} : \{0,1\}^{n_{\mathsf{ENC}}} \times \{0,1\}^{n_{\mathsf{ENC}}} \to \{0,1\}^{n_{\mathsf{ENC}}}$ *be an ideal block-cipher. Let* F *be* $(2, q_G, q_{1,F}, t_1, \epsilon_{\text{2-upL-KE}})$-2-upL-KE-*secure with the key generated by* π^{KE}, $(2, q_G, q'_{1,F}, t'_1, \epsilon_{\text{2-upL}})$-2-upL *unpredictable with leakage, and be* $(2, q_G, q_{2,F}, t_2, \epsilon_{\text{2-upL-\$XORL}})$-2-upL-\$XORL-*secure, and* $(2, q_G, q_{2,F}, t_2, \epsilon_{\text{2-upL-\$XORL}})$-2-upL-\$XOR2L-*secure with the key randomly sampled and XORed to a random value. Let* ENC *be* $(q_{3,\mathsf{F}}, t_3, \epsilon_{\text{EavL-GenFL}})$-EavL-GenFL *with the key picked via* F, *then, for any* (q_{F}, t) *adversary* $\mathsf{A} = (\mathsf{A}_1^{\mathsf{F}}, \mathsf{A}_2^{\mathsf{F}})$-*adversary,* $\pi^{N\text{-LR-OT}}$ *is* $(q_{\mathsf{F}}, t, \epsilon)$-OT-S-LA *secure, with*

$$\epsilon \leq \epsilon_{\text{2-upL-KE}} + 2\epsilon_{\text{2-upL-\$XORL}} + 2N\epsilon_{\text{EavL-GenFL}} \\ +2(N-1)(\epsilon_{\text{2-upL}} + \epsilon_{\text{2-upL-\$XOR2L}})$$

$t_1 = t + N(t_{\mathsf{OT}} + 2t_{\mathsf{F}} + 2t_{\mathsf{Enc}}) + 4(N-1)t_{\mathsf{F}}$, $t'_1 = t + t_{\mathsf{KE}} + N(t_{\mathsf{OT}} + 2t_{\mathsf{F}} + 2t_{\mathsf{Enc}}) + (4N-2)t_{\mathsf{F}}$, $t_2 = t + N(t_{\mathsf{OT}} + 2t_{\mathsf{F}} + 2t_{\mathsf{Enc}}) + (4N-2)t_{\mathsf{F}}$, $t_3 = t + N(t_{\mathsf{OT}} + 2t_{\mathsf{F}}) + (N-1)(2t_{\mathsf{F}} + t_{\mathsf{Enc}})$, $q_{1,F} = 2 + q_{\mathsf{F}}$, $q_{2,F} = 1 + q_{\mathsf{F}}$, *and* $q_{3,F} = q_{\mathsf{F}}$.

Proof. The proof is inspired by those in the same model for F in [9].

Hybrids. We use a sequence of Hybrids, Hybrid 0, Hybrid $0^1, \ldots$ Hybrid 0^8 where an adversary A is playing. Let E_i be the event that the adversary A outputs 1 at the end of Hybrid i.

- **Hybrid 0:** It is the OT-S-LA Hybrid where A is playing against $\pi^{\mathsf{LR\text{-}OT}}$, where the bit picked is 0.
- **Hybrid 0^1:** It is Hybrid 0, where we replace $z_{1,0}, z_{1,1}$ with random values, and the leakage of F accordingly.
- **Hybrid 0^2:** It is Hybrid 0^1, where we replace $y_{1,0}, k_{1,0,E}, k_{1,0,S}$ with three random values and their leakage accordingly.
- **Hybrid 0^3:** It is Hybrid 0^2, where we replace $y_{1,1}, k_{1,1,E}, k_{1,1,S}$ with three random values and their leakage accordingly.
- **Hybrid 0^4:** It is Hybrid 0^3, where we replace $c_{1,0}$ with $\mathsf{Enc}_{k_{1,0,E}}(m^1_{1,0})$ (that is, instead of encrypting $m^0_{1,0}$, we encrypt $m^1_{1,0}$), and its leakage accordingly.
- **Hybrid 0^5:** It is Hybrid 0^4, where we replace $c_{1,1}$ with $\mathsf{Enc}_{k_{1,1,E}}(m^1_{1,1})$ (that is, instead of encrypting $m^0_{1,1}$, we encrypt $m^1_{1,1}$), and its leakage accordingly.
- **Hybrid 0^6 (Complementing 0^3):** It is Hybrid 0^5, where we replace $y_{1,1}$, $k_{1,1,E}$, and $k_{1,1,S}$ with the correct values, and the leakage accordingly.
- **Hybrid 0^7 (Complementing 0^2):** It is Hybrid 0^6, where we replace $y_{1,0}$, $k_{1,0,E}$, and $k_{1,0,S}$ with the correct values, and the leakage accordingly.
- **Hybrid 0^8 (Complementing 1):** It is Hybrid 0^7, where we replace $z_{1,0}$, $z_{1,1}$ with the correct values, and the leakage accordingly.
  We call Hybrid 1, Hybrid 0^8.
- Then, we have a sequence of Hybrids, Hybrid $1^1, \ldots$ Hybrid 1^{10}, Hybrid $2^1, \ldots$, Hybrid N^{10}:
  - **Hybrid i^1:** It is Hybrid i, where we replace $z_{i,00}, z_{i,01}$ with random values, and the leakage of F accordingly.
  - **Hybrid i^2:** It is Hybrid i^i, where we replace $z_{i,10}, z_{i,11}$ with random values, and the leakage of F accordingly.
  - **Hybrid i^3:** It is Hybrid i^2, where we replace $y_{i,00}, y_{i,10}, k_{i,0,E}, k_{i,0,S}$ with four random values and their leakage accordingly.

- **Hybrid i^4:** It is Hybrid i^3, where we replace $y_{i,01}, y_{i,11}, k_{i,1,E}, k_{i,1,S}$ with four random values and their leakage accordingly.
- **Hybrid i^5:** It is Hybrid i^4, where we replace $c_{i,0}$ with $\mathsf{Enc}_{k_{i,0,E}}(m^1_{i,0})$ (that is, instead of encrypting $m^0_{i,0}$, we encrypt $m^1_{i,0}$), and its leakage accordingly.
- **Hybrid i^6:** It is Hybrid 0^5, where we replace $c_{i,1}$ with $\mathsf{Enc}_{k_{i,1,E}}(m^1_{i,1})$ (that is, instead of encrypting $m^0_{i,1}$, we encrypt $m^1_{i,1}$), and its leakage accordingly.
- **Hybrid i^7 (Complementing i^4):** It is Hybrid i^5, where we replace $y_{i,10}$,$y_{i,11}$, $k_{i,1,E}$, and $k_{i,1,S}$ with the correct values, and the leakage accordingly.
- **Hybrid i^8 (Complementing i^3):** It is Hybrid i^7, where we replace $y_{i,00}$, $y_{i,01}$, $k_{i,0,E}$, and $k_{i,0,S}$ with the correct values, and the leakage accordingly.
- **Hybrid i^9 (Complementing i^2):** It is Hybrid i^8, where we replace $z_{i,10}$, $z_{i,11}$ with the correct values, and the leakage accordingly.
- **Hybrid i^{10} (Complementing i^1):** It is Hybrid i^9, where we replace $z_{i,00}$, $z_{i,01}$ with the correct values, and the leakage accordingly.
  We call Hybrid $i+1$, Hybrid i^{10}.

Indistinguishability of Hybrids. Each of the previous Hybrids is indistinguishable from the following as we now prove:

- **From Hybrid 0 to 0^1:** We observe the two Hybrids are the same except if the adversary A does a query to F with the key k because it would cause the loss of randomness of $y_{1,0}$, and $y_{1,1}$ [59]. Thus,
$$|\Pr[E_0] - \Pr[E_{0^1}]| \leq \epsilon_{\mathsf{2\text{-}upL\text{-}KE}}.$$
- **From Hybrid 0^1 to 0^2:** We observe the two Hybrids are the same except if the adversary A does a query to F with the key $k_{1,0}$ because it would cause the loss of randomness of $y_{1,0}$ or $k_{1,0,E}$ or $k_{1,0,S}$. Thus,
$$|\Pr[E_{0^1}] - \Pr[E_{0^2}]| \leq \epsilon_{\mathsf{2\text{-}upL\text{-}\$XORL}}.$$
- **From Hybrid 0^2 to 0^3:** We observe the two Hybrids are the same except if the adversary A does a query to F with the key $k_{1,1}$ because it would cause the loss of randomness of $y_{1,1}$ or $k_{1,1,E}$ or $k_{1,1,S}$. Thus,
$$|\Pr[E_{0^2}] - \Pr[E_{0^3}]| \leq \epsilon_{\mathsf{2\text{-}upL\text{-}\$XORL}}.$$
- **From Hybrid 0^3 to 0^4:** Since ENC is $(q_{3,\mathsf{F}}, t_3, \epsilon_{\mathsf{EavL\text{-}GenFL}})$ and the result of Lemma 1
$$|\Pr[E_3] - \Pr[E_4]| \leq 2\epsilon_{\mathsf{EavL\text{-}GenFL}}.$$
- **From Hybrid 0^4 to 0^5:** Similarly, as before
$$|\Pr[E_4] - \Pr[E_5]| \leq 2\epsilon_{\mathsf{EavL\text{-}GenFL}}.$$
- **From Hybrid 0^5 to 0^8:** It is the indistinguishiability of Hybrids 0, 0^1, 0^2, and 0^3 done in the opposite order. Thus,
$$|\Pr[E_{0^5}] - \Pr[E_{0^8}]| \leq \epsilon_{\mathsf{2\text{-}upL\text{-}KE}} + 2\epsilon_{\mathsf{2\text{-}upL\text{-}\$XORL}}.$$
- For $i = 1, \ldots, N-1$

- **From Hybrid i to i^1:** We observe the two Hybrids are the same except if the adversary A does a query to F with the key $k_{i-1,0,S}$ because it would cause the loss of randomness of $y_{i,00}$, and $y_{i,01}$ [59]. Thus,
$$|\Pr[E_{i^0}] - \Pr[E_{i^1}]| \le \epsilon_{\mathsf{2\text{-}upL}}.$$
- **From Hybrid i^1 to i^2:** We observe the two Hybrids are the same except if the adversary A does a query to F with the key $k_{i-1,1,S}$ because it would cause the loss of randomness of $y_{i,10}$, and $y_{i,11}$ [59]. Thus,
$$|\Pr[E_{i^1}] - \Pr[E_{i^2}]| \le \epsilon_{\mathsf{2\text{-}upL}}.$$
- **From Hybrid i^2 to i^3:** We observe the two Hybrids are the same except if the adversary A does a query to F with the key $k_{i,0}$ because it would cause the loss of randomness of $y_{i,00}$, $y_{i,10}$ or $k_{i,0,E}$ or $k_{i,0,S}$. Thus,
$$|\Pr[E_{i^2}] - \Pr[E_{i^3}]| \le \epsilon_{\mathsf{2\text{-}upL\text{-}\$XOR2L}}.$$
- **From Hybrid i^3 to i^4:** We observe the two Hybrids are the same except if the adversary A does a query to F with the key $k_{i,1}$ because it would cause the loss of randomness of $y_{i,01}$, $y_{i,11}$ or $k_{i,1,E}$ or $k_{i,1,S}$. Thus,
$$|\Pr[E_{i^2}] - \Pr[E_{i^3}]| \le \epsilon_{\mathsf{2\text{-}upL\text{-}\$XOR2L}}.$$
- **From Hybrid i^4 to i^5:** Since ENC is $(q_{3,\mathsf{F}}, t_3, \epsilon_{\mathsf{EavL\text{-}GenFL}})$ and the result of Lemma 1
$$|\Pr[E_{i^4}] - \Pr[E_{i^5}]| \le 2\epsilon_{\mathsf{EavL\text{-}GenFL}}.$$
- **From Hybrid i^5 to i^6:** Similarly, as before
$$|\Pr[E_{i^5}] - \Pr[E_{i^6}]| \le 2\epsilon_{\mathsf{EavL\text{-}GenFL}}.$$
- **From Hybrid i^6 to i^{10}:** It is the indistinguishiability of Hybrids i, i^1, i^2, i^3 and i^4 done in the opposite order. Thus,
$$|\Pr[E_{i^5}] - \Pr[E_{i^{10}}]| \le 2(\mathsf{2\text{-}upL} + 2\epsilon_{\mathsf{2\text{-}upL\text{-}\$XOR2L}}).$$

Now, we observe that Hybrid 0 is the $\mathsf{OT\text{-}S\text{-}LA}$ game where the bit picked is 0, while Hybrid N is the $\mathsf{OT\text{-}S\text{-}LA}$ game where the bit is picked 1. Thus,
$$|\Pr[E_0] - \Pr[E_N]| \le$$
$$4N\epsilon_{\mathsf{EavL\text{-}GenFL}} + 4(N-1)\epsilon_{\mathsf{2\text{-}upL\text{-}\$XOR2L}} + 4(N-1)\epsilon_{\mathsf{2\text{-}upL}} + 4\epsilon_{\mathsf{2\text{-}upL\text{-}\$XORL}} + 2\epsilon_{\mathsf{2\text{-}upL\text{-}KE}}.$$
Using Lemma 1
$$|\Pr[\mathsf{A}\text{ wins}] - 1/2| \le 1/2(\Pr[E_0] - \Pr[E_N]) \le$$
$$\epsilon_{\mathsf{2\text{-}upL\text{-}KE}} + 2\epsilon_{\mathsf{2\text{-}upL\text{-}\$XORL}} + 2N\epsilon_{\mathsf{EavL\text{-}GenFL}} + 2(N-1)[\epsilon_{\mathsf{2\text{-}upL}} + \epsilon_{\mathsf{2\text{-}upL\text{-}\$XOR2L}}].$$
□

5.3 OT-S-LR -Security of $\pi^{N\text{-}\mathsf{LR\text{-}OT}}$

The proof for corrupted adversaries relies, in addition to the previous modifications, on the fact that $y_{i,\bar{r}_{i-1}r_i}$, being equal to $\mathsf{F}_{k_{i-1,\bar{r}_{i-1},S}}(P_{r_i}) \oplus k_{i,r_i}$, gives no information to the receiver, since it should not be able to guess $k_{i-1,\bar{r}_{i-1},S}$. We now give the details.

First, we have to adapt the composition definition for F, \$, $\oplus$ function, and OT (Def. 25) to the case where 2 $\oplus$ functions are computed:

Definition 27. *Let* π^{OT} *be an* OT *protocol,* $\$$ *be the random sampling,* $\oplus$ *be the XOR function, and* $\mathsf{L}^{\mathsf{S}}_{\mathsf{OT}}(\cdot,\cdot)$*,* $\mathsf{L}_{\$}(x)$*, and* $\mathsf{L}_{\oplus}(x,y)$ *be the leakage of their respective implementations. Let* $\mathsf{F} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ *be an ideal cipher and* L_{F} *be its leakage function. Let* $\$$ *generate* k*, which is the key used by the oracle* FL*. Let* k *be XORed to an adversarial chosen value* z*, obtaining* $y = (x \oplus k)$*. Let* y *be the* $y_{\bar{r}}$ *input of the* π^{OT} *protocol with* r *and* y_r *chosen by the adversary. We say that the implementation of the composition of the random sampling, the XOR,* π^{OT}*, and* F *is* $(q, q_G, q_{\mathsf{F}}, t, \epsilon)$-unpredictable in the presence of leakage $\mathsf{L}_{\$}(x), \mathsf{L}_{\oplus}(x,y), \mathsf{L}_{\mathsf{F}}(\cdot;\cdot), \mathsf{L}^{\mathsf{S}}_{\mathsf{OT}}(\cdot,\cdot)$ with π^{OT} using a random input*, denoted as* q-upL-$\$\mathsf{XOR2L}$-$\mathsf{OT}$*, if for any* (q, q_{F}, t)*-adversary* $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$*,*

$$\begin{aligned}
&\Pr[k \in \mathcal{G} \mid |\mathcal{G}| \leq q_G,\\
&\mathcal{G} \leftarrow \mathsf{A}_2^{\mathsf{FL}_k(\cdot),\mathsf{F}.(\cdot)}[\mathsf{st}, \mathsf{L}_{\$}(k), \mathsf{L}_{\oplus}(z_0,k), \mathsf{L}_{\oplus}(z_1,k),\\
&\mathsf{pp}^{\mathsf{OT}}, \mathsf{trans}_{\mathsf{OT}}, \mathsf{L}^{\mathsf{S}}_{\mathsf{OT}}((y_{00},y_{10}),(y_{01},y_{11}),r)],\\
&(\mathsf{pp}^{\mathsf{OT}}, \mathsf{trans}_{\mathsf{OT}}, \mathsf{L}^{\mathsf{S}}_{\mathsf{OT}}((y_{00},y_{10}),(y_{01},y_{10}),r) \leftarrow \pi^{\mathsf{OT}}\mathsf{L}(1^{\kappa}((y_{00},y_{10}),(y_{01},y_{10}),r),\\
&y_{0r} \leftarrow y_0,\ y_{1r} \leftarrow y_1,\ y_{0\bar{r}} = z_0 \oplus k,\ y_{1\bar{r}} = z_1 \oplus k\ (k, \mathsf{L}_{\$}(k)) \xleftarrow{\$} \{0,1\}^n,\\
&(\mathsf{st}, z_0, z_1, r, y_0, y_1) \leftarrow \mathsf{A}_1^{\mathsf{F}.(\cdot)}] \leq \epsilon,
\end{aligned}$$

with q_{F} *and the oracle* F *defined as in Def. 6.*

In our case $q = 1$.

Theorem 6. *Let* $\mathsf{F} : \{0,1\}^{n_{\mathsf{ENC}}} \times \{0,1\}^{n_{\mathsf{ENC}}} \to \{0,1\}^{n_{\mathsf{ENC}}}$ *be an ideal block-cipher, and* $(2, q_G, q_{1,F}, t_1, \epsilon_{2\text{-}\mathsf{upL}})$-2-$\mathsf{upL}$ *unpredictable with leakage, Let* F *be* $(2, q_G, q_{1,F}, t_1, \epsilon_{2\text{-}\mathsf{upL}\text{-}\$\mathsf{XORL}\text{-}\mathsf{OT}})$ -2-upL*-secure, and* F *be* $(2, q_G, q_{1,F}, t_1, \epsilon_{2\text{-}\mathsf{upL}\text{-}\$\mathsf{XOR2L}\text{-}\mathsf{OT}})$ -2-upL*-secure, with the key randomly sampled and XORed to a random value and* π^{OT} *sending the XOR. Let* ENC *be* $(q_{3,\mathsf{F}}, t_3, \epsilon_{\mathsf{EavL}\text{-}\mathsf{GenFL}})$-$\mathsf{EavL}$-$\mathsf{GenFL}$ *with the key is picked via* F*, then, for any* (q_{F}, t) *adversary* $\mathsf{A} = (\mathsf{A}_1^{\mathsf{F}}, \mathsf{A}_2^{\mathsf{F}})$*-adversary,* $\pi^{N\text{-}\mathsf{LR}\text{-}\mathsf{OT}}$ *is* $(q_{\mathsf{F}}, t, \epsilon)$-$\mathsf{OT}$-$\mathsf{S}$-$\mathsf{LR}$ *secure, with*

$$\epsilon \leq \epsilon_{2\text{-}\mathsf{upL}\text{-}\$\mathsf{XORL}\text{-}\mathsf{OT}} + N(\epsilon_{2\text{-}\mathsf{upL}} + \epsilon_{\mathsf{EavL}\text{-}\mathsf{GenFL}}) + (N-1)(\epsilon_{2\text{-}\mathsf{upL}} + \epsilon_{2\text{-}\mathsf{upL}\text{-}\$\mathsf{XOR2L}\text{-}\mathsf{OT}})$$

with $t_1 = t + t_{\mathsf{KE}} + N(t_{\mathsf{OT}} + 2t_{\mathsf{F}} + 2t_{\mathsf{Enc}}) + (4N-2)t_{\mathsf{F}}$*,* $t_2 = t + N(t_{\mathsf{OT}} + 2t_{\mathsf{F}} + 2t_{\mathsf{Enc}}) + (4N-2)t_{\mathsf{F}}$*,* $t_3 = t + N(t_{\mathsf{OT}} + 2t_{\mathsf{F}}) + (N-1)(2t_{\mathsf{F}} + t_{\mathsf{Enc}})$*,* $q_{1,F} = 2 + q_{\mathsf{F}}$*,* $q_{2,F} = 1 + q_{\mathsf{F}}$*, and* $q_{3,F} = q_{\mathsf{F}}$*.*

Proof. The proof is inspired by those in the same model for F in [9].

Hybrids. We use a sequence of Hybrids, Hybrid 0, ..., Hybrid 3 where an adversary A is playing. Let E_i be the event that the adversary A outputs 1 at the end of Hybrid i.

- **Hybrid 0:** It is the OT-S-LR Hybrid where A is playing against $\pi^{\mathsf{LR}\text{-}\mathsf{OT}}$, where the bit picked is 0.
- **Hybrid 0^1:** It is Hybrid 0, where we replace $y_{1,\bar{r}_1}, k_{1,\bar{r}_1,E}, k_{1,\bar{r}_1,S}$ with three random values and their leakage accordingly.
- **Hybrid 0^2:** It is Hybrid 0^1, where we replace $c_{1,\bar{r}_1}$ with $\mathsf{Enc}_{k_{1,\bar{r}_1,E}}(m^1_{1,\bar{r}_1})$ (that is, instead of encrypting $m^0_{1,\bar{r}_1}$, we encrypt $m^1_{1,\bar{r}_1}$), and its leakage accordingly.

– **Hybrid** 0^3**:** It is Hybrid 0^2, where we replace $y_{1,\bar{r}_1}$, $k_{1,\bar{r}_1,E}$ and $k_{1,\bar{r}_1,S}$ with the correct values, and the leakage accordingly.

– Then, we have a sequence of Hybrids, Hybrid 1^1, ... Hybrid 1^5, Hybrid 2^1, ..., Hybrid N^5:
  - **Hybrid** i^1**:** It is Hybrid i, where we replace $z_{i,\bar{r}_{i-1}0}, z_{i,\bar{r}_{i-1}1}$ with two random values and their leakage accordingly.
  - **Hybrid** i^2**:** It is Hybrid i^1, where we replace $y_{i,0\bar{r}_i}, y_{i,1\bar{r}_i}, k_{i,\bar{r}_i,E}, k_{i,\bar{r}_i,S}$ with four random values and their leakage accordingly.
  - **Hybrid** i^3**:** It is Hybrid i^2, where we replace $c_{i,\bar{r}_i}$ with $\mathsf{Enc}_{k_{i,\bar{r}_i,E}}(m^1_{i,\bar{r}_i})$ (that is, instead of encrypting $m^0_{i,\bar{r}_i}$, we encrypt $m^1_{i,\bar{r}_i}$), and its leakage accordingly.
  - **Hybrid** i^4**:** It is Hybrid i^3, where we replace $y_{i,0\bar{r}_i}, y_{i,1\bar{r}_i}$, $k_{i,\bar{r}_i,E}$ and $k_{i,\bar{r}_i,S}$ with the correct values, and the leakage accordingly.
  - **Hybrid** i^5**:** It is Hybrid i^4, where we replace $z_{i,\bar{r}_{i-1}0}, z_{i,\bar{r}_{i-1}1}$ w w$z_{i,\bar{r}_{i-1}0}, z_{i,\bar{r}_{i-1}1}$ with the correct values, and the leakage accordingly.
    We call Hybrid i^4, Hybrid $i+1$.

Indistinguishability of Hybrids. Each of the previous Hybrids is indistinguishable from the following as we now prove:

– **From Hybrid 0 to** 0^1**:** We observe the two Hybrids are the same except if the adversary A does a query to F with the key $k_{1,\bar{r}_1}$ because it would cause the loss of randomness of $k_{1,\bar{r}_1,E}$, and $k_{1,\bar{r}_1,S}$. Thus,

$$|\Pr[E_0] - \Pr[E_{0^1}]| \leq \epsilon_{\mathsf{2\text{-}upL\text{-}\$XORL\text{-}OT}}.$$

– **From Hybrid** 0^1 **to** 0^2**:** Since ENC is $(q_{3,\mathsf{F}}, t_3, \epsilon_{\mathsf{EavL\text{-}GenFL}})$ and the result of Lemma 1

$$|\Pr[E_{0^1}] - \Pr[E_{0^2}]| \leq 2\epsilon_{\mathsf{EavL\text{-}GenFL}}.$$

– **From Hybrid** 0^2 **to** 0^3**:** It is the transition between Hybrids 0 and 0^1 done in the opposite order. Thus,

$$|\Pr[E_{0^2}] - \Pr[E_{0^3}]| \leq \epsilon_{\mathsf{2\text{-}upL\text{-}\$XORL\text{-}OT}}.$$

– For $i = 1, \ldots N-1$
  - **From Hybrid 0 to** 0^1**:** We observe the two Hybrids are the same except if the adversary A does a query to F with the key $k_{i-1,\bar{r}_{i-1},S}$ because it would cause the loss of randomness of $z_{i,\bar{r}_{i-1}0}$, and $z_{i,\bar{r}_{i-1}1}$. Thus,

$$|\Pr[E_i] - \Pr[E_{i^1}]| \leq \epsilon_{\mathsf{2\text{-}upL}}.$$

  - **From Hybrid** i^1 **to** i^2**:** We observe the two Hybrids are the same except if the adversary A does a query to F with the key $k_{i,\bar{r}_i}$ because it would cause the loss of randomness of $y_{i,0\bar{r}_i}, y_{i,1\bar{r}_i}, k_{i,\bar{r}_i,E}$, and $k_{i,\bar{r}_i,S}$. Thus,

$$|\Pr[E_{i^1}] - \Pr[E_{i^2}]| \leq \epsilon_{\mathsf{2\text{-}upL\text{-}\$XOR2L\text{-}OT}}.$$

  - **From Hybrid** i^2 **to** i^3**:** Since ENC is $(q_{3,\mathsf{F}}, t_3, \epsilon_{\mathsf{EavL\text{-}GenFL}})$ and the result of Lemma 1

$$|\Pr[E_{i^2}] - \Pr[E_{i^3}]| \leq 2\epsilon_{\mathsf{EavL\text{-}GenFL}}.$$

  - **From Hybrid** i^3 **to** i^5**:** They are the transitions between Hybrids i and i^1, and between i^1 and i^2, done in the opposite order. Thus,

$$|\Pr[E_{i^3}] - \Pr[E_{i^5}]| \leq \epsilon_{\mathsf{2\text{-}upL\text{-}\$XOR2L\text{-}OT}} + \epsilon_{\mathsf{2\text{-}upL}}.$$

Now, we observe that Hybrid 0 is the $\mathsf{OT\text{-}S\text{-}LR}$ game where the bit picked is 0, while Hybrid N is the $\mathsf{OT\text{-}S\text{-}LR}$ game where the bit is picked 1. Thus, $|\Pr[E_0]-\Pr[E_N]| \leq 2N\epsilon_{\mathsf{EavL\text{-}GenFL}}+2\epsilon_{\mathsf{2\text{-}upL\text{-}\$XORL\text{-}OT}}+2(N-1)(\epsilon_{\mathsf{2\text{-}upL\text{-}\$XOR2L\text{-}OT}}+\epsilon_{\mathsf{2\text{-}upL}})$. Using Lemma 1

$$|\Pr[\mathsf{A}\text{ wins}] - 1/2| \leq 1/2(\Pr[E_0] - \Pr[E_N]) \leq$$
$$\epsilon_{\mathsf{2\text{-}upL\text{-}\$XORL\text{-}OT}} + N\epsilon_{\mathsf{EavL\text{-}GenFL}} + (N-1)(\epsilon_{\mathsf{2\text{-}upL\text{-}\$XOR2L\text{-}OT}} + \epsilon_{\mathsf{2\text{-}upL}}).$$

□

6 Variants

In this section, we briefly discuss variants of our protocols that require stronger security requirements but achieve better efficiency and additional properties.

1. **No refresh:** We avoid Step 2c (and Step 1c for $\pi^{N\text{-}\mathsf{LR\text{-}OT}}$), and use k_0, k_1 as keys for ENC (in $\pi^{\mathsf{LR\text{-}OT}}$, Step 2d), and $k_{i,j}$ as keys for ENC (in $\pi^{N\text{-}\mathsf{LR\text{-}OT}}$, Step 1d). We describe these changes in the single instance protocol, $\pi^{\mathsf{LR\text{-}OTf}}$ (Alg. 3), and in the sequential protocol, $\pi^{N\text{-}\mathsf{LR\text{-}OTf}}$ (Alg. 4). This requires relying on a more aggressive hypothesis about the following combination because the same key is involved in the random sampling, the XOR function, the F computation, the OT computation, and the Enc computation. That is, we have to combine Def 22 and Def. 23 for intercepting adversaries, and Def 25 and Def. 23 for corrupted receivers. This change can yield a faster protocol but requires using the keys more times.
2. **Using the inverse of F:** we can skip the use of the $\oplus$ in step 2b by computing $y_0 = \mathsf{F}_k(k_0)$ and $y_1 = \mathsf{F}_k(k_1)$, instead of $y_i = \mathsf{F}_k(P_i) \oplus k_i$ for $\pi^{\mathsf{LR\text{-}OT}}$. Thus, in Def. 22 we have to replace the leakage $\mathsf{L}_{\oplus}(z_i, k_i)$ by the leakage $\mathsf{L}_{\mathsf{F}}(k; k_i)$. We describe these changes in the single instance protocol (Alg. 5), and in the sequential protocol (Alg. 6). We consider this option essentially equivalent in terms of efficiency and security and as a flavor that allows flexibility in implementation.
3. **Tree composition of multiple instances**: the idea is to generate the keys $k_{i,j,S}$ in Step 1f, instead of Step 2b. In particular, $k_{i,j}$ is used to generate two different keys, $k_{2i,j,S}$ and $k_{2i+1,S}$. This way, the $2i^{\text{th}}$ and the $(2i+1)^{\text{th}}$ instances can be run in parallel (Alg. 7). This requires more evaluations of F, but it allows the protocol to be executed in $O(\log(N))$ rounds, thus trading additional calls F with a reduced number of rounds.
4. **Parallel composition of multiple instances:** Here we use k to encrypt all keys $k_{i,j}$, thus, we can execute all instances in parallel. (Alg. 8). Since k is used $2N$ times by F, we must assume that F is leak-free. This construction requires a stronger hypothesis on the block-cipher in the presence of leakage, where the main added value is a faster scheme with high performance.

Acknowledgments

Francesco Berti and Itamar Levi were founded by the Israel Science Foundation (ISF) grant 2569/21. Carmit Hazay was partially supported by the Algorand

Centres of Excellence programme managed by Algorand Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Algorand Foundation, and the United States-Israel Binational Science Foundation (BSF) through Grant No. 2020277.

References

1. A. Akavia, S. Goldwasser, and V. Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC*, 2009.
2. G. Barwell, D. P. Martin, E. Oswald, and M. Stam. Authenticated encryption in the face of protocol and side channel leakage. In *ASIACRYPT*, pages 693–723, 2017.
3. D. Beaver. Precomputing oblivious transfer. In *CRYPTO*, 1995.
4. D. Beaver. Correlated pseudorandomness and the complexity of private computations. In *STOC*, 1996.
5. M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *EUROCRYPT*, 2006.
6. D. Bellizia, F. Berti, O. Bronchain, G. Cassiers, S. Duval, C. Guo, G. Leander, G. Leurent, I. Levi, C. Momin, O. Pereira, T. Peters, F. Standaert, B. Udvarhelyi, and F. Wiemer. Spook: Sponge-based leakage-resistant authenticated encryption with a masked tweakable block cipher. *IACR Trans. Symmetric Cryptol.*, (S1):295–349, 2020.
7. D. Bellizia, O. Bronchain, G. Cassiers, V. Grosso, C. Guo, C. Momin, O. Pereira, T. Peters, and F. Standaert. Mode-level vs. implementation-level physical security in symmetric cryptography - A practical guide through the leakage-resistance jungle. In *CRYPTO*, pages 369–400, 2020.
8. F. Berti, C. Guo, O. Pereira, T. Peters, and F. Standaert. Strong authenticity with leakage under weak and falsifiable physical assumptions. In *Inscrypt*, 2019.
9. F. Berti, C. Guo, O. Pereira, T. Peters, and F. Standaert. Tedt, a leakage-resist AEAD mode for high physical security applications. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, (1):256–320, 2020.
10. F. Berti, C. Guo, T. Peters, Y. Shen, and F. Standaert. Secure message authentication in the presence of leakage and faults. *IACR Trans. Symmetric Cryptol.*, 2023(1):288–315, 2023.
11. N. Bitansky, R. Canetti, and S. Halevi. Leakage-tolerant interactive protocols. In *TCC*, 2012.
12. E. A. Bock, C. Brzuska, W. Michiels, and A. Treff. On the ineffectiveness of internal encodings - revisiting the DCA attack on white-box cryptography. In *ACNS*, 2018.
13. A. Bogdanov, Y. Ishai, and A. Srinivasan. Unconditionally secure computation against low-complexity leakage. In *CRYPTO*, 2019.
14. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *SIGSAC*, 2019.
15. E. Boyle, S. Goldwasser, A. Jain, and Y. T. Kalai. Multiparty computation secure against continual memory leakage. In *STOC*, 2012.
16. G. Cassiers, B. Grégoire, I. Levi, and F.-X. Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Transactions on Computers*, 70(10):1677–1690, 2020.

17. J. Coron, J. Patarin, and Y. Seurin. The random oracle model and the ideal cipher model are equivalent. In *CRYPTO*, pages 1–20, 2008.
18. N. Costes and M. Stam. Pincering SKINNY by exploiting slow diffusion enhancing differential power analysis with cluster graph inference. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023.
19. G. Couteau, P. Rindal, and S. Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In *CRYPTO*, 2021.
20. J. P. Degabriele, C. Janson, and P. Struck. Sponges resist leakage: The case of authenticated encryption. In *ASIACRYPT*, pages 209–240, 2019.
21. C. Dobraunig, M. Eichlseder, S. Mangard, F. Mendel, and T. Unterluggauer. ISAP - towards side-channel secure authenticated encryption. *IACR Trans. Symmetric Cryptol.*, (1):80–105, 2017.
22. Y. Dodis and J. P. Steinberger. Message authentication codes from unpredictable block ciphers. In *CRYPTO*, 2009.
23. S. Dziembowski and K. Pietrzak. Leakage-resilient cryptography. In *FOCS*, 2008.
24. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. In *CRYPTO*, 1982.
25. O. Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. 2004.
26. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, 1987.
27. C. Guo, O. Pereira, T. Peters, and F. Standaert. Authenticated encryption with nonce misuse and physical leakage: Definitions, separation results and first construction. In *LATINCRYPT*, pages 150–172, 2019.
28. J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *USENIX*, 2008.
29. C. Hazay and Y. Lindell. *Efficient Secure Two-Party Protocols - Techniques and Constructions*. Information Security and Cryptography. 2010.
30. R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *STOC*.
31. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *CRYPTO*, 2003.
32. Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, 2008.
33. K. Järvinen and J. Balasch. Single-trace side-channel attacks on scalar multiplications with precomputations. In *Smart Card Research and Advanced Applications: 15th International Conference, CARDIS 2016, Cannes, France, November 7–9, 2016, Revised Selected Papers 15*, pages 137–155. Springer, 2017.
34. I. Kabin, Z. Dyka, D. Klann, and P. Langendoerfer. Horizontal attacks against ecc: from simulations to asic. In *Computer Security: ESORICS 2019 International Workshops, IOSec, MSTEC, and FINSEC, Luxembourg City, Luxembourg, September 26–27, 2019, Revised Selected Papers 2*, pages 64–76. Springer, 2020.
35. Y. T. Kalai and L. Reyzin. A survey of leakage-resilient cryptography. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. ACM, 2019.
36. J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. 2014.
37. J. Kilian. Founding cryptography on oblivious transfer. In *STOC*, 1988.
38. P. C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, pages 104–113, 1996.

39. P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *CRYPTO*, pages 388–397, 1999.
40. I. Levi and C. Hazay. Garbled circuits from an SCA perspective free XOR can be quite expensive. . *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023.
41. J. Longo, D. P. Martin, E. Oswald, D. Page, M. Stam, and M. Tunstall. Simulatable leakage: Analysis, pitfalls, and new constructions. In *ASIACRYPT*, pages 223–242, 2014.
42. S. Micali and L. Reyzin. Physically observable cryptography. In *TCC*, pages 278–296, 2004.
43. C. Namprempre, P. Rogaway, and T. Shrimpton. Reconsidering generic composition. In *EUROCRYPT*, 2014.
44. C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, 2008.
45. O. Pereira, F. Standaert, and S. Vivek. Leakage-resilient authentication and encryption from symmetric cryptographic primitives. In *CCS*, pages 96–108, 2015.
46. J. Quisquater and D. Samyde. Electromagnetic analysis (EMA): measures and counter-measures for smart cards. In *E-smart*, 2001.
47. M. O. Rabin. How to exchange secrets with oblivious transfer. Technical report, Harvard Center for Research in Computer Technology, 1981.
48. S. Raghuraman, P. Rindal, and T. Tanguy. Expand-convolute codes for pseudorandom correlation generators from LPN. In *CRYPTO*, 2023.
49. P. Ravi, R. Poussier, S. Bhasin, and A. Chattopadhyay. On configurable sca countermeasures against single trace attacks for the ntt: A performance evaluation study over kyber and dilithium on the arm cortex-m4. In *Security, Privacy, and Applied Cryptography Engineering: 10th International Conference, SPACE 2020, Kolkata, India, December 17–21, 2020, Proceedings 10*, pages 123–146. Springer, 2020.
50. L. Roy. Softspokenot: Communication-computation tradeoffs in OT extension. *IACR Cryptol. ePrint Arch.*, page 192, 2022.
51. P. S. Roy and A. Adhikari. One-sided leakage-resilient privacy only two-message oblivious transfer. *J. Inf. Secur. Appl.*, pages 295–300, 2014.
52. D. Salomon and I. Levi. Masksimd-lib: on the performance gap of a generic c optimized assembly and wide vector extensions for masked software with an ascon-p test case. *Journal of Cryptographic Engineering*, 13(3):325–342, 2023.
53. D. Salomon, A. Weiss, and I. Levi. Improved filtering techniques for single-and multi-trace side-channel analysis. *Cryptography*, 5(3):24, 2021.
54. M. Staib and A. Moradi. Deep learning side-channel collision attack. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023.
55. Y. Tang, Z. Gong, J. Chen, and N. Xie. Higher-order DCA attacks on white-box implementations with masking and shuffling countermeasures. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023.
56. S. Wolf and J. Wullschleger. Oblivious transfer is symmetric. In *EUROCRYPT*, 2006.
57. A. C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, 1986.
58. S. You, M. G. Kuhn, S. Sarkar, and F. Hao. Low trace-count template attacks on 32-bit implementations of ASCON AEAD. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023.
59. Y. Yu, F. Standaert, O. Pereira, and M. Yung. Practical leakage-resilient pseudorandom generators. In *CCS*, pages 141–151, 2010.

A Algorithms

Algorithm 3 Our $\pi^{\mathsf{LR\text{-}OTf}}$ protocol for a single OT instance. With respect to the π^{OT} protocol, we have skipped Step 2c.

- **Building blocks:**
  1. $\pi^{\mathsf{KE}} = (\pi^{\mathsf{KE}}.\mathsf{Setup}, \pi^{\mathsf{KE}}.\mathsf{KE})$, a key-exchange protocol,
  2. F, a block-cipher,
  3. $\mathsf{ENC} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, an EavL-secure encryption scheme,
  4. $\pi^{\mathsf{OT}} = (\pi^{\mathsf{OT}}.\mathsf{Setup}, \pi^{\mathsf{OT}}.\mathsf{OT})$, an OT protocol.
- **Input:** S has a couple of strings $M := (m_0, m_1)$, with $|m_0| = |m_1|$; R has a bit r
- **Auxiliary input:** 1^κ the security parameter (shared by both S and R)
- **Setup phase:** $\pi^{\mathsf{LR\text{-}OT}}.\mathsf{Setup}(1^\kappa)$ does:
  1. From 1^κ choose n_{ENC}, a BC $\mathsf{F} : \{0,1\}^{n_{\mathsf{ENC}}} \times \{0,1\}^{n_{\mathsf{ENC}}} \to \{0,1\}^{n_{\mathsf{ENC}}}$, an EavL-secure encryption scheme ENC, two different strings $P_0, P_1 \in \{0,1\}^{n_{\mathsf{ENC}}}$.
  2. $\mathsf{pp}^{\mathsf{KE}} \leftarrow \pi^{\mathsf{KE}}.\mathsf{Setup}(1^\kappa, n_{\mathsf{ENC}})$,
  3. $\mathsf{pp}^{\mathsf{OT}} \leftarrow \pi^{\mathsf{OT}}.\mathsf{Setup}(1^\kappa)$
  4. Return $\mathsf{pp} = (\mathsf{pp}^{\mathsf{KE}}, \mathsf{pp}^{\mathsf{OT}}, n_{\mathsf{ENC}}, \mathsf{F}, \mathsf{ENC}, P_0, P_1, P_2)$
- **Main phase:** $\pi^{\mathsf{LR\text{-}OTf}}.\mathsf{OT}(\mathsf{pp})$:
  1. **Key-exchange phase**: S and R execute $\pi^{\mathsf{KE}}.\mathsf{KE}(\mathsf{pp}^{\mathsf{KE}})$ to both obtain k
  2. **Sender's phase:**
     (a) S picks $k_0, k_1 \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$
     (b) S computes $z_0 = \mathsf{F}_k(P_0),\ y_0 = z_0 \oplus k_0,\ z_1 = \mathsf{F}_k(P_1),\ y_1 = z_1 \oplus k_1,$
     (c) S computes $c_0 = \mathsf{Enc}_{k_0}(m_0),\ c_1 = \mathsf{Enc}_{k_1}(m_1)$
     (d) S sends c_0, c_1 to R
  3. **Send-key phase:** S and R execute $\pi^{\mathsf{OT}}.\mathsf{OT}(\mathsf{pp}^{\mathsf{OT}})$ with S's input (y_0, y_1) and R's r
  4. **Receiver's phase:**
     R computes $k_{\mathsf{R}} = \mathsf{F}_k(P_r) \oplus y_r$,
     R computes $m = \mathsf{Dec}_{k_{\mathsf{R}}}(c_r)$
- **Output**: S: nothing; R: m

Algorithm 4 Our N-OT protocol $\pi^{N\text{-}\mathsf{LR\text{-}OTf}} = (\pi^{N\text{-}\mathsf{LR\text{-}OT}}.\mathsf{Setup}, \pi^{N\text{-}\mathsf{LR\text{-}OTf}}.\mathsf{OT})$. With respect to $\pi^{N\text{-}\mathsf{LR\text{-}OT}}$ we skip step 2c for the first OT instance, and step 2d) in all other instances. Moreover, in all other instances in step 2b) we skip the computation of $k_{i,j,S}$.

- **Building blocks:**
  1. $\pi^{\mathsf{KE}} = (\pi^{\mathsf{KE}}.\mathsf{Setup}, \pi^{\mathsf{KE}}.\mathsf{KE})$, a key-exchange protocol,
  2. F, a block-cipher,
  3. $\mathsf{ENC} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, an EavL-secure encryption scheme,
  4. $\pi^{\mathsf{OT}} = (\pi^{\mathsf{OT}}.\mathsf{Setup}, \pi^{\mathsf{OT}}.\mathsf{OT})$, an OT protocol.
- **Input:** S has n-couples of strings $M := ((m_{1,0}, m_{1,1}), ..., (m_{N,0}, m_{N,1}))$ with $|m_{i,0}| = |m_{i,1}|$ $\forall i \in [N]$; R has a string $r = (r_1, ..., r_N) \in \{0,1\}^N$
- **Auxiliary input:** 1^κ the security parameter (shared by both S and R)
- **Setup phase:** $\pi^{\mathsf{LR\text{-}OT}}.\mathsf{Setup}(1^\kappa)$ does:
  1. From 1^κ choose n_{ENC}, a BC $\mathsf{F} : \{0,1\}^{n_{\mathsf{ENC}}} \times \{0,1\}^{n_{\mathsf{ENC}}} \to \{0,1\}^{n_{\mathsf{ENC}}}$, an EavL-secure encryption scheme ENC, four different strings $P_0, P_1, P_2, P_3 \in \{0,1\}^{n_{\mathsf{ENC}}}$.
  2. $\mathsf{pp}^{\mathsf{KE}} \leftarrow \pi^{\mathsf{KE}}.\mathsf{Setup}(1^\kappa, n_{\mathsf{ENC}})$,
  3. $\mathsf{pp}^{\mathsf{OT}} \leftarrow \pi^{\mathsf{OT}}.\mathsf{Setup}(1^\kappa)$
  4. Return $\mathsf{pp} = (\mathsf{pp}^{\mathsf{KE}}, \mathsf{pp}^{\mathsf{OT}}, n_{\mathsf{ENC}}, \mathsf{F}, \mathsf{ENC}, P_0, P_1, P_2)$
- $\pi^{\mathsf{LR\text{-}OT}}.\mathsf{OT}(\mathsf{pp})$:
- **First OT instance:**
  1. **Key-exchange phase**: S and R execute $\pi^{\mathsf{KE}}.\mathsf{KE}(\mathsf{pp}^{\mathsf{KE}})$ to both obtain k
  2. **Sender's phase:**
     (a) S picks $k_{1,0}, k_{1,1} \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$
     (b) S computes $z_{1,0} = \mathsf{F}_k(P_0)$, $y_{1,0} = z_{1,0} \oplus k_{1,0}$, $z_{1,1} = \mathsf{F}_k(P_1)$, $y_{1,1} = z_{1,1} \oplus k_{1,1}$,
     (c) S computes $c_{1,0} = \mathsf{Enc}_{k_{1,0}}(m_{1,0})$, $c_{1,1} = \mathsf{Enc}_{k_{1,1}}(m_{1,1})$
     (d) S sends $c_{1,0}, c_{1,1}$ to R
  3. **Send-key phase:** S and R execute $\pi^{\mathsf{OT}}.\mathsf{OT}(\mathsf{pp}^{\mathsf{OT}})$ with S's input $(y_{1,0}, y_{1,1})$ and R's r_1
  4. **Receiver's phase:**
     R computes $k_{\mathsf{R},1} = \mathsf{F}_k(P_r) \oplus y_{1,r_1}$
     R computes $m_1 = \mathsf{Dec}_{k_{\mathsf{R},1}}(c_{1,r_1})$
- **Remaining OT instances:** For $i = 2, \dots, N$
  1. **Sender's phase:**
     (a) S picks $k_{i,0}, k_{i,1} \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$
     (b) S computes $z_{i,00} = \mathsf{F}_{k_{i-1,0}}(P_2)$, $y_{i,00} = z_{i,00} \oplus k_{i,0}$, $z_{i,01} = \mathsf{F}_{k_{i-1,0}}(P_3)$, $y_{i,01} = z_{i,01} \oplus k_{i,1}$, $z_{i,10} = \mathsf{F}_{k_{i-1,1}}(P_2)$, $y_{i,10} = z_{i,10} \oplus k_{i,0}$, $z_{i,11} = \mathsf{F}_{k_{i-1,1}}(P_3)$, $y_{i,11} = z_{i,11} \oplus k_{i,11}$
     (c) S computes $c_{i,0} = \mathsf{Enc}_{k_{i,0}}(m_{i,0})$, $c_{i,1} = \mathsf{Enc}_{k_{i,1}}(m_{i,1})$
     (d) S sends $c_{i,0}, c_{i,1}$ to R
  2. **Send-key phase:** S and R execute $\pi^{\mathsf{OT}}.\mathsf{OT}(\mathsf{pp}^{\mathsf{OT}})$ with S's input $((y_{i,00}, y_{i,10}), (y_{i,01}, y_{i,11})$ and R's r_i
  3. **Receiver's phase:**
     R computes $k_{\mathsf{R},i} = \mathsf{F}_{k_{\mathsf{R},i-1}}(P_{2+r_i}) \oplus y_{i,r_{i-1}r_i}$
     R computes $m_i = \mathsf{Dec}_{k_{\mathsf{R},i}}(c_{i,r_i})$
- **Output**: S: nothing; R: $M_{\mathsf{R}} = (m_1, ..., m_N)$

Algorithm 5 Our $\pi^{\textsf{LR-OTinv}}$ protocol for a single $\textsf{OT}$ instance. With respect to $\pi^{\textsf{LR-OT}}$, we have modified Step 2b.

- **Building blocks:**
  1. $\pi^{\textsf{KE}} = (\pi^{\textsf{KE}}.\textsf{Setup}, \pi^{\textsf{KE}}.\textsf{KE})$, a key-exchange protocol,
  2. $\textsf{F}$, a block-cipher,
  3. $\textsf{ENC} = (\textsf{Gen}, \textsf{Enc}, \textsf{Dec})$, an $\textsf{EavL}$-secure encryption scheme,
  4. $\pi^{\textsf{OT}} = (\pi^{\textsf{OT}}.\textsf{Setup}, \pi^{\textsf{OT}}.\textsf{OT})$, an $\textsf{OT}$ protocol.
- **Input:** $\textsf{S}$ has a couple of strings $M := (m_0, m_1)$, with $|m_0| = |m_1|$; $\textsf{R}$ has a bit r
- **Auxiliary input:** 1^κ the security parameter (shared by both $\textsf{S}$ and $\textsf{R}$)
- **Setup phase:** $\pi^{\textsf{LR-OT}}.\textsf{Setup}(1^\kappa)$ does:
  1. From 1^κ choose $n_{\textsf{ENC}}$, a $\textsf{BC}$ $\textsf{F} : \{0,1\}^{n_{\textsf{ENC}}} \times \{0,1\}^{n_{\textsf{ENC}}} \to \{0,1\}^{n_{\textsf{ENC}}}$, an $\textsf{EavL}$-secure encryption scheme $\textsf{ENC}$, three different strings $P_0, P_1, P_2 \in \{0,1\}^{n_{\textsf{ENC}}}$.
  2. $\textsf{pp}^{\textsf{KE}} \leftarrow \pi^{\textsf{KE}}.\textsf{Setup}(1^\kappa, n_{\textsf{ENC}})$,
  3. $\textsf{pp}^{\textsf{OT}} \leftarrow \pi^{\textsf{OT}}.\textsf{Setup}(1^\kappa)$
  4. Return $\textsf{pp} = (\textsf{pp}^{\textsf{KE}}, \textsf{pp}^{\textsf{OT}}, n_{\textsf{ENC}}, \textsf{F}, \textsf{ENC}, P_0, P_1, P_2)$
- **Main phase:** $\pi^{\textsf{LR-OTinv}}.\textsf{OT}(\textsf{pp})$:
  1. **Key-exchange phase**: $\textsf{S}$ and $\textsf{R}$ execute $\pi^{\textsf{KE}}.\textsf{KE}(\textsf{pp}^{\textsf{KE}})$ to both obtain k
  2. **Sender's phase:**
     (a) $\textsf{S}$ picks $k_0, k_1 \overset{\$}{\leftarrow} \{0,1\}^{n_{\textsf{ENC}}}$
     (b) $\textsf{S}$ computes $y_0 = \textsf{F}_k(k_0),\ y_1 = \textsf{F}_k(k_1)$,
     (c) $\textsf{S}$ computes $k_{0,E} = \textsf{F}_{k_0}(P_2)$, $k_{1,E} = \textsf{F}_{k_1}(P_2)$,
     (d) $\textsf{S}$ computes $c_0 = \textsf{Enc}_{k_{0,E}}(m_0),\ c_1 = \textsf{Enc}_{k_{1,E}}(m_1)$
     (e) $\textsf{S}$ sends c_0, c_1 to $\textsf{R}$
  3. **Send-key phase:** $\textsf{S}$ and $\textsf{R}$ execute $\pi^{\textsf{OT}}.\textsf{OT}(\textsf{pp}^{\textsf{OT}})$ with $\textsf{S}$'s input (y_0, y_1) and $\textsf{R}$'s r
  4. **Receiver's phase:**
     $\textsf{R}$ computes $k_{\textsf{R}} = \textsf{F}_k^{-1}(y_r)$, $k_{\textsf{R},E} = \textsf{F}_{k_{\textsf{R}}}(P_2)$,
     $\textsf{R}$ computes $m = \textsf{Dec}_{k_{\textsf{R},E}}(c_r)$
- **Output**: $\textsf{S}$: nothing; $\textsf{R}$: m

Algorithm 6 Our N-OT protocol $\pi^{N\text{-}\mathsf{LR}\text{-}\mathsf{OTinv}} = (\pi^{N\text{-}\mathsf{LR}\text{-}\mathsf{OT}}.\mathsf{Setup}, \pi^{N\text{-}\mathsf{LR}\text{-}\mathsf{OTinv}}.\mathsf{OT})$. With respect to $\pi^{N\text{-}\mathsf{LR}\text{-}\mathsf{OT}}$, we have modified Step 2b for the first instance and Step 1b for all other instances

```
– Building blocks:
    1. $\pi^{\mathsf{KE}} = (\pi^{\mathsf{KE}}.\mathsf{Setup}, \pi^{\mathsf{KE}}.\mathsf{KE})$, a key-exchange protocol,
    2. $\mathsf{F}$, a block-cipher,
    3. $\mathsf{ENC} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, an $\mathsf{EavL}$-secure encryption scheme,
    4. $\pi^{\mathsf{OT}} = (\pi^{\mathsf{OT}}.\mathsf{Setup}, \pi^{\mathsf{OT}}.\mathsf{OT})$, an $\mathsf{OT}$ protocol.
– Input: $\mathsf{S}$ has $n$-couples of strings $M := ((m_{1,0}, m_{1,1}), ..., (m_{N,0}, m_{N,1}))$ with
  $|m_{i,0}| = |m_{i,1}|\ \forall i \in [N]$; $\mathsf{R}$ has a string $r = (r_1, ..., r_N) \in \{0,1\}^N$
– Auxiliary input: $1^\kappa$ the security parameter (shared by both $\mathsf{S}$ and $\mathsf{R}$)
– Setup phase: $\pi^{\mathsf{LR}\text{-}\mathsf{OT}}.\mathsf{Setup}(1^\kappa)$ does:
    1. From $1^\kappa$ choose $n_{\mathsf{ENC}}$, a $\mathsf{BC}$ $\mathsf{F} : \{0,1\}^{n_{\mathsf{ENC}}} \times \{0,1\}^{n_{\mathsf{ENC}}} \to \{0,1\}^{n_{\mathsf{ENC}}}$, an $\mathsf{EavL}$-
       secure encryption scheme $\mathsf{ENC}$, four different strings $P_0, P_1, P_2, P_3 \in \{0,1\}^{n_{\mathsf{ENC}}}$.
    2. $\mathsf{pp}^{\mathsf{KE}} \leftarrow \pi^{\mathsf{KE}}.\mathsf{Setup}(1^\kappa, n_{\mathsf{ENC}})$,
    3. $\mathsf{pp}^{\mathsf{OT}} \leftarrow \pi^{\mathsf{OT}}.\mathsf{Setup}(1^\kappa)$
    4. Return $\mathsf{pp} = (\mathsf{pp}^{\mathsf{KE}}, \mathsf{pp}^{\mathsf{OT}}, n_{\mathsf{ENC}}, \mathsf{F}, \mathsf{ENC}, P_0, P_1, P_2)$
– $\pi^{N\text{-}\mathsf{LR}\text{-}\mathsf{OTinv}}.\mathsf{OT}(\mathsf{pp})$:
– First OT instance:
    1. Key-exchange phase: $\mathsf{S}$ and $\mathsf{R}$ execute $\pi^{\mathsf{KE}}.\mathsf{KE}(\mathsf{pp}^{\mathsf{KE}})$ to both obtain $k$
    2. Sender's phase:
        (a) $\mathsf{S}$ picks $k_{1,0}, k_{1,1} \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$
        (b) $\mathsf{S}$ computes $y_{1,0} = \mathsf{F}_k(k_{1,0}),\ y_{1,1} = \mathsf{F}_k(k_{1,1})$,
        (c) $\mathsf{S}$ computes $k_{1,0,E} = \mathsf{F}_{k_{1,0}}(P_2)$, $k_{1,1,E} = \mathsf{F}_{k_{1,1}}(P_2)$,
        (d) $\mathsf{S}$ computes $c_{1,0} = \mathsf{Enc}_{k_{1,0,E}}(m_{1,0}),\ c_{1,1} = \mathsf{Enc}_{k_{1,1,E}}(m_{1,1})$
        (e) $\mathsf{S}$ sends $c_{1,0}, c_{1,1}$ to $\mathsf{R}$
    3. Send-key phase: $\mathsf{S}$ and $\mathsf{R}$ execute $\pi^{\mathsf{OT}}.\mathsf{OT}(\mathsf{pp}^{\mathsf{OT}})$ with $\mathsf{S}$'s input $(y_{1,0}, y_{1,1})$
       and $\mathsf{R}$'s $r_1$
    4. Receiver's phase:
         $\mathsf{R}$ computes $k_{\mathsf{R},1} = \mathsf{F}_k^{-1}(y_{1,r})$, $k_{\mathsf{R},1,E} = \mathsf{F}_{k_{\mathsf{R},1}}(P_2)$,
         $\mathsf{R}$ computes $m_1 = \mathsf{Dec}_{k_{\mathsf{R},1,E}}(c_{1,r_1})$
– Remaining OT instances: For $i = 2, \ldots, N$
    1. Sender's phase:
        (a) $\mathsf{S}$ picks $k_{i,0}, k_{i,1} \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$
        (b) $\mathsf{S}$ computes $k_{i-1,0,S} = \mathsf{F}_{k_{i-1,0}}(P_3)$, $k_{i-1,1,S} = \mathsf{F}_{k_{i-1,1}}(P_3)$, computes
            $y_{i,00} = \mathsf{F}_{k_{i-1,0,S}}(k_{i,0})$, $y_{i,01} = \mathsf{F}_{k_{i-1,0,S}}(k_{i,1})$, $y_{i,10} = \mathsf{F}_{k_{i-1,1,S}}(k_{i,0})$,
            $y_{i,11} = \mathsf{F}_{k_{i-1,1,S}}(k_{i,1})$
        (c) $\mathsf{S}$ computes $k_{i,0,E} = \mathsf{F}_{k_{i,0}}(P_2)$, $k_{i,1,E} = \mathsf{F}_{k_{i,1}}(P_2)$,
        (d) $\mathsf{S}$ computes $c_{i,0} = \mathsf{Enc}_{k_{i,0,E}}(m_{i,0}),\ c_{i,1} = \mathsf{Enc}_{k_{i,1,E}}(m_{i,1})$
        (e) $\mathsf{S}$ sends $c_{i,0}, c_{i,1}$ to $\mathsf{R}$
    2. Send-key phase: $\mathsf{S}$ and $\mathsf{R}$ execute $\pi^{\mathsf{OT}}.\mathsf{OT}(\mathsf{pp}^{\mathsf{OT}})$ with $\mathsf{S}$'s input
       $((y_{i,00}, y_{i,10}), (y_{i,01}, y_{i,11}))$ and $\mathsf{R}$'s $r_i$
    3. Receiver's phase:
         $\mathsf{R}$ computes $k_{\mathsf{R},i-1,S} = \mathsf{F}_{k_{\mathsf{R},i-1}}(P_3)$, $k_{\mathsf{R},i} = \mathsf{F}^{-1}_{k_{\mathsf{R},i-1,S}}(y_{i,r_{i-1}r_i})$, $k_{\mathsf{R},i,E} =$
            $\mathsf{F}_{k_{\mathsf{R},i}}(P_2)$,
         $\mathsf{R}$ computes $m_i = \mathsf{Dec}_{k_{\mathsf{R},i,E}}(c_{i,r_i})$
– Output: $\mathsf{S}$: nothing; $\mathsf{R}$: $M_{\mathsf{R}} = (m_1, ..., m_N)$
```

Algorithm 7 Our N-OT protocol $\pi^{N\text{-}\mathsf{LR\text{-}OTtree}} = (\pi^{N\text{-}\mathsf{LR\text{-}OT}}.\mathsf{Setup}, \pi^{N\text{-}\mathsf{LR\text{-}OTtree}}.\mathsf{OT})$. With respect to $\pi^{N\text{-}\mathsf{LR\text{-}OT}}$, we have added Step 2f for the first instance and Step 1f for all other instances. In this phase, we compute the sending keys for two instances.

- **Building blocks:** As for $\pi^{N\text{-}\mathsf{LR\text{-}OT}}$.
- **Input:** S has n-couples of strings $M := ((m_{1,0}, m_{1,1}), ..., (m_{N,0}, m_{N,1}))$ with $|m_{i,0}| = |m_{i,1}|\ \forall i \in [N]$; R has a string $r = (r_1, ..., r_N) \in \{0,1\}^N$
- **Auxiliary input:** 1^κ the security parameter (shared by both S and R)
- **Setup phase:** As for $\pi^{N\text{-}\mathsf{LR\text{-}OT}}$.
- $\pi^{N\text{-}\mathsf{LR\text{-}OTtree}}.\mathsf{OT}(\mathsf{pp})$:
- **First OT instance:**
  1. **Key-exchange phase**: S and R execute $\pi^{\mathsf{KE}}.\mathsf{KE}(\mathsf{pp}^{\mathsf{KE}})$ to both obtain k
  2. **Sender's phase:**
     (a) S picks $k_{1,0}, k_{1,1} \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$
     (b) S computes $z_{1,0} = \mathsf{F}_k(P_0)$, $y_{1,0} = z_{1,0} \oplus k_{1,0}$, $z_{1,1} = \mathsf{F}_k(P_1)$, $y_{1,1} = z_{1,1} \oplus k_{1,1}$,
     (c) S computes $k_{1,0,E} = \mathsf{F}_{k_{1,0}}(P_2)$, $k_{1,1,E} = \mathsf{F}_{k_{1,1}}(P_2)$,
     (d) S computes $c_{1,0} = \mathsf{Enc}_{k_{1,0,E}}(m_{1,0})$, $c_{1,1} = \mathsf{Enc}_{k_{1,1,E}}(m_{1,1})$
     (e) S sends $c_{1,0}, c_{1,1}$ to R
     (f) S computes $k_{1,0,Ref} = \mathsf{F}_{k_{1,0}}(P_3)$, $k_{1,1,Ref} = \mathsf{F}_{k_{1,1}}(P_3)$. S computes $k_{2,0,S} = \mathsf{F}_{k_{1,0,Ref}}(P_0)$, $k_{2,1,S} = \mathsf{F}_{k_{1,1,Ref}}(P_0)$, $k_{3,0,S} = \mathsf{F}_{k_{1,0,Ref}}(P_1)$, $k_{3,1,S} = \mathsf{F}_{k_{1,1,Ref}}(P_1)$.
  3. **Send-key phase:** S and R execute $\pi^{\mathsf{OT}}.\mathsf{OT}(\mathsf{pp}^{\mathsf{OT}})$ with S's input $(y_{1,0}, y_{1,1})$ and R's r_1
  4. **Receiver's phase:**
     R computes $k_{\mathsf{R},1} = \mathsf{F}_k(P_r) \oplus y_{1,r_1}$, $k_{\mathsf{R},1,E} = \mathsf{F}_{k_{\mathsf{R},1}}(P_2)$,
     R computes $m_1 = \mathsf{Dec}_{k_{\mathsf{R},1,E}}(c_{1,r_1})$
     R computes $k_{\mathsf{R},1,Ref} = \mathsf{F}_{k_{\mathsf{R},1}}(P_3)$, $k_{\mathsf{R},2,S} = \mathsf{F}_{k_{\mathsf{R},1,Ref}}(P_0)$, $k_{\mathsf{R},3,S} = \mathsf{F}_{k_{\mathsf{R},1,Ref}}(P_1)$.
- **Remaining OT instances:** For $i = 2, \ldots, N$
  1. **Sender's phase:**
     (a) S picks $k_{i,0}, k_{i,1} \overset{\$}{\leftarrow} \{0,1\}^{n_{\mathsf{ENC}}}$
     (b) S computes $z_{i,00} = \mathsf{F}_{k_{i,0,S}}(P_0)$, $y_{i,00} = z_{i,00} \oplus k_{i,0}$, $z_{i,01} = \mathsf{F}_{k_{i,0,S}}(P_1)$, $y_{i,01} = z_{i,01} \oplus k_{i,1}$, $z_{i,10} = \mathsf{F}_{k_{i,1,S}}(P_0)$, $y_{i,10} = z_{i,10} \oplus k_{i,0}$, $z_{i,11} = \mathsf{F}_{k_{i,1,S}}(P_1)$, $y_{i,11} = z_{i,11} \oplus k_{i,11}$
     (c) S computes $k_{i,0,E} = \mathsf{F}_{k_{i,0}}(P_2)$, $k_{i,1,E} = \mathsf{F}_{k_{i,1}}(P_2)$,
     (d) S computes $c_{i,0} = \mathsf{Enc}_{k_{i,0,E}}(m_{i,0})$, $c_{i,1} = \mathsf{Enc}_{k_{i,1,E}}(m_{i,1})$
     (e) S sends $c_{i,0}, c_{i,1}$ to R
     (f) S computes $k_{i,0,Ref} = \mathsf{F}_{k_{i,0}}(P_3)$, $k_{i,1,Ref} = \mathsf{F}_{k_{i,1}}(P_3)$. S computes $k_{2i,0,S} = \mathsf{F}_{k_{i,0,Ref}}(P_0)$, $k_{2i,1,S} = \mathsf{F}_{k_{i,1,Ref}}(P_0)$, $k_{2i+1,0,S} = \mathsf{F}_{k_{i,0,Ref}}(P_1)$, $k_{2i+1,1,S} = \mathsf{F}_{k_{i,1,Ref}}(P_1)$.
  2. **Send-key phase:** S and R execute $\pi^{\mathsf{OT}}.\mathsf{OT}(\mathsf{pp}^{\mathsf{OT}})$ with S's input $((y_{i,00}, y_{i,10}), (y_{i,01}, y_{i,11})$ and R's r_i
  3. **Receiver's phase:**
     R computes $k_{\mathsf{R},i} = \mathsf{F}_{k_{\mathsf{R},i,S}}(P_{r_i}) \oplus y_{i,r_{i-1}r_i}$, $k_{\mathsf{R},i,E} = \mathsf{F}_{k_{\mathsf{R},i}}(P_2)$,
     R computes $m_i = \mathsf{Dec}_{k_{\mathsf{R},i,E}}(c_{i,r_i})$
     R computes $k_{\mathsf{R},i,Ref} = \mathsf{F}_{k_{\mathsf{R},i}}(P_3)$, $k_{\mathsf{R},2i,S} = \mathsf{F}_{k_{\mathsf{R},i,Ref}}(P_0)$, $k_{\mathsf{R},2i+1,S} = \mathsf{F}_{k_{\mathsf{R},i,Ref}}(P_1)$.
- **Output**: S: nothing; R: $M_{\mathsf{R}} = (m_1, ..., m_N)$

Algorithm 8 Our N-OT protocol $\pi^{N\text{-LR-OTpar}} = (\pi^{N\text{-LR-OT}}.\mathsf{Setup}, \pi^{N\text{-LR-OTpar}}.\mathsf{OT})$. With respect to $\pi^{N\text{-LR-OT}}$, we use k to encrypt all ephemeral keys $k_{i,j}$. (With $i\|j$ we denote the string obtained writing the number i as a $n_{\mathsf{ENC}} - 1$ bit long string, and appending the bit j)

- **Building blocks:**
  1. $\pi^{\mathsf{KE}} = (\pi^{\mathsf{KE}}.\mathsf{Setup}, \pi^{\mathsf{KE}}.\mathsf{KE})$, a key-exchange protocol,
  2. F, a block-cipher,
  3. $\mathsf{ENC} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, an EavL-secure encryption scheme,
  4. $\pi^{\mathsf{OT}} = (\pi^{\mathsf{OT}}.\mathsf{Setup}, \pi^{\mathsf{OT}}.\mathsf{OT})$, an OT protocol.
- **Input:** S has n-couples of strings $M := ((m_{1,0}, m_{1,1}), ..., (m_{N,0}, m_{N,1}))$ with $|m_{i,0}| = |m_{i,1}|\ \forall i \in [N]$; R has a string $r = (r_1, ..., r_N) \in \{0,1\}^N$
- **Auxiliary input:** 1^κ the security parameter (shared by both S and R)
- **Setup phase:** $\pi^{\mathsf{LR\text{-}OT}}.\mathsf{Setup}(1^\kappa)$ does:
  1. From 1^κ choose n_{ENC}, a BC $\mathsf{F} : \{0,1\}^{n_{\mathsf{ENC}}} \times \{0,1\}^{n_{\mathsf{ENC}}} \to \{0,1\}^{n_{\mathsf{ENC}}}$, an EavL-secure encryption scheme ENC, a string $P_2 \in \{0,1\}^{n_{\mathsf{ENC}}}$.
  2. $\mathsf{pp}^{\mathsf{KE}} \leftarrow \pi^{\mathsf{KE}}.\mathsf{Setup}(1^\kappa, n_{\mathsf{ENC}})$,
  3. $\mathsf{pp}^{\mathsf{OT}} \leftarrow \pi^{\mathsf{OT}}.\mathsf{Setup}(1^\kappa)$
  4. Return $\mathsf{pp} = (\mathsf{pp}^{\mathsf{KE}}, \mathsf{pp}^{\mathsf{OT}}, n_{\mathsf{ENC}}, \mathsf{F}, \mathsf{ENC}, P_0, P_1, P_2)$
- $\pi^{\mathsf{LR\text{-}OT}}.\mathsf{OT}(\mathsf{pp})$:
  1. **Key-exchange phase**: S and R execute $\pi^{\mathsf{KE}}.\mathsf{KE}(\mathsf{pp}^{\mathsf{KE}})$ to both obtain k
  2. For $i = 1, \dots, N$
     - i **Sender's phase:**
       - (a) S picks $k_{i,0}, k_{i,1} \xleftarrow{\$} \{0,1\}^{n_{\mathsf{ENC}}}$
       - (b) S computes $z_{i,0} = \mathsf{F}_k(i\|0)$, $y_{i,0} = z_{i,0} \oplus k_{i,0}$, $z_{i,1} = \mathsf{F}_k(i\|1)$, $y_{i,1} = z_{i,1} \oplus k_{i,1}$,
       - (c) S computes $k_{i,0,E} = \mathsf{F}_{k_{i,0}}(P_2)$, $k_{i,1,E} = \mathsf{F}_{k_{i,1}}(P_2)$,
       - (d) S computes $c_{i,0} = \mathsf{Enc}_{k_{i,0,E}}(m_{i,0})$, $c_{i,1} = \mathsf{Enc}_{k_{i,1,E}}(m_{i,1})$
       - (e) S sends $c_{i,0}, c_{i,1}$ to R
     - ii **Send-key phase:** S and R execute $\pi^{\mathsf{OT}}.\mathsf{OT}(\mathsf{pp}^{\mathsf{OT}})$ with S's input $(y_{i,0}, y_{i,1})$ and R's r_i
     - iii **Receiver's phase:**
       R computes $k_{\mathsf{R},i} = \mathsf{F}_k(i\|r_i) \oplus y_{i,r_i}$, $k_{\mathsf{R},i,E} = \mathsf{F}_{k_{\mathsf{R},i}}(P_2)$,
       R computes $m_i = \mathsf{Dec}_{k_{\mathsf{R},i,E}}(c_{i,r_i})$
- **Output**: S: nothing; R: $M_{\mathsf{R}} = (m_1, ..., m_N)$