Conditionally Input-Revealing 2PC and Fuzzy Password-Authenticated Key Exchange*

David Richardson Mike Rosulek Jiayu Xu

February 24, 2026

Abstract

Yao's famous protocol for secure 2-party computation, based on garbled circuits, is well-known to be insecure against an actively corrupt garbler. We introduce a new and extremely simple variant of Yao's protocol that is fully secure against active adversaries, for a certain class of functions that we call *conditionally input-revealing.*

We then show how to use this new protocol as the basis for fuzzy password authenticated key exchange (fuzzy PAKE). In fuzzy PAKE, two parties each hold a low-entropy secret (e.g., a password), and they interact to obtain a secure high-entropy key if and only if the passwords are sufficiently "close." Our new fuzzy PAKE protocol supports completely arbitrary predicates for password "closeness." Compared to prior fuzzy PAKE protocols, ours is roughly 2× cheaper in communication, computation, and round complexity.

1 Introduction

Background: Active security of Yao's protocol. Yao's protocol [32] is the first protocol for secure multi-party computation, and still a leading paradigm almost 40 years later. Yao's protocol provides security against semi-honest adversaries, and even against an actively corrupt receiver. However, it is insecure against an actively corrupt sender, because such an adversary may garble a circuit other than the agreed-upon one, to reveal too much information about the receiver's input.

Many variants of Yao's protocol have been proposed, achieving security against an actively corrupt sender. In the cut-and-choose approach [20, 21, 30, 19, 26], the sender generates many copies of a garbled circuit, some of which are checked (opened) for correctness, while the others are evaluated as usual. In the dual-execution approach [25, 14, 15, 18], parties run two copies of Yao's protocol, in opposite directions, so that both parties have the opportunity to act as sender and receiver. The most basic variants of this approach achieve security against active adversaries, but leak a single adversarially-chosen bit about the inputs. The authenticated garbling approach [31, 17] uses a completely different distributed approach to generating the underlying garbled circuit, to prevent the sender from cheating.

Fuzzy PAKE. In standard password-authenticated key exchange (PAKE) [4], two parties each hold a low-entropy password string. By running the PAKE protocol, the parties are able to obtain a secure shared (high-entropy) key in the presence of an eavesdropper, if and only if the passwords match. Thus, PAKE allows parties to bootstrap from low-entropy shared secrets to high-entropy shared secrets during an authentication phase.

A natural variant of PAKE is called *fuzzy PAKE* [9]. Here, the passwords do not need to match completely; they merely need to be "close enough." Fuzzy PAKE can be useful as a way to tolerate innocuous password typos (and thereby improve usability) [8], or as an essential part of biometric authentication, where noisy biometric measurements are inevitable.

*Oregon State University, {richdavi,rosulekm,xujiay}@oregonstate.edu. First two authors partially supported by NSF award 2150726.

1.1 Our Results

Conditionally input-revealing 2PC. A **conditionally input-revealing (CIR)** ideal functionality is parameterized by a predicate Π, and has the following behavior (informally):

> Await input x from the receiver and input y from the sender. If $\Pi(x, y) = \texttt{true}$ then give y to the receiver, else give $\perp$ to the receiver. (Give no output to the sender.)

We propose a new and simple protocol for securely evaluating CIR functionalities. The protocol has essentially the same cost as Yao's (semi-honest) protocol evaluation of Π, yet is secure against actively corrupt adversaries. We use a novel twist on Yao's protocol which may be of independent interest (see the overview below).

Fuzzy PAKE. When Π is a predicate that measures whether two passwords are "close," our CIR protocol becomes the core mechanism of a protocol for **fuzzy PAKE**. Intuitively, suppose the sender's password is pw and it chooses a random K and runs our CIR protocol on input (pw, K). The receiver's input to the CIR protocol is their password pw'. If $\Pi(pw, pw') = 0$ (the passwords are not sufficiently close), the protocol reveals nothing to the receiver. If $\Pi(pw, pw') = 1$, the protocol reveals the sender's input to the receiver, including K. Thus, the parties agree on K if and only if their passwords are close.

Comparison to existing fuzzy PAKE. There are relatively few fuzzy PAKE protocols, which are summarized in Table 1. Dupont et al. (DHPRY) [9] introduced fuzzy PAKE and proposed two protocols. Their first protocol is based on garbled circuits and Yao's protocol, like ours. They overcome the semi-honest security of Yao's protocol by running two parallel instances in opposite directions, in the *dual execution* paradigm [25]. Usually, dual execution leaks an adversarially chosen bit about the honest party's input, but DHPRY cleverly arrange things so that the bit is leaked only when the passwords are sufficiently close. Our protocol uses only one instance of Yao's protocol instead of two, and adds only an additional commitment to the sender's guess (and a short seed).

The second protocol of DHPRY is an optimized variant supporting Hamming distance similarity only. This protocol was later shown to be insecure by [6], so we do not include it in our comparison.

Bootle et al. (BFHHO) [6] describe a fuzzy PAKE that supports Hamming distance similarity. It involves running a separate instance of (exact) PAKE for each character of the password. Then, using error correcting codes with efficient list decoding, the parties can safely arrange a common key when enough of the PAKE instances succeed — i.e., when enough of the password characters match. In the event of a successful match, the protocol leaks the locations of the matching characters to one of the parties.

Both of the existing protocols use the "split authentication" compiler of [1], which adds two rounds to the beginning of the protocol (to establish signing keys and consensus about those keys) and adds signatures to every protocol message. We give a more direct construction that avoids this compiler, and saves 2 protocol rounds.

1.2 Other Related Work

Our protocol has the curious feature that a single garbled circuit is evaluated and then later opened. Something similar also appears in the zero-knowledge proof protocol of Jawurek, Kerschbaum, and Orlandi (JKO) [16]. However, the two techniques are significantly different beyond this high-level description. In the JKO protocol, the garbler has no input to the circuit. Thus, they can use a garbling scheme which does not have the traditional privacy property [11]; only the authenticity property is needed.

In contrast, the garbler in our protocol does have input to the circuit, so we do need the garbling scheme to provide privacy. We destroy the privacy guarantee *conditionally,* depending on whether the evaluator is entitled to learn the garbler's private input.

In this work, we consider *symmetric* fuzzy PAKE, where both parties hold a password in the clear. Other works have incorporated "fuzziness" into *asymmetric* variants of PAKE [10, 28, 24], where one party (a client) holds the password in the clear, and the other (a server) holds only a one-way digest of the password.

	DHPRY [9]	BFHHO [6]	**ours**
similarity predicate	any	Hamming	any
if passwords match, receiver (R) learns sender (S) learns	 entire password entire password	 matching chars nothing	 entire password nothing
communication rounds	R ⇌ ⇌ ⇌ ⇌ S	R ⇌ ⇌ ⇌ ← S	R → ← S
approximate cost	$2 \times$ Yao	$n \times$ PAKE	$1 \times$ Yao

Table 1: Comparison of fuzzy PAKE protocols. "⇌" indicates a message from each party that can be sent simultaneously, while "→" and "←" indicate a message from just one party. Red arrows indicate the "split authentication" compiler of [1]. n is the number of password characters.

Our work also has similarities to the Fujisaki-Okamoto (FO) transform [12] for public-key encryption. In the FO transform, the receiver learns the sender's (purported) randomness, then re-encrypts the plaintext with that randomness and compares the result to the ciphertext that was received. Our protocol can be viewed as an application of the FO transform to Yao's protocol: The receiver learns the garbler's randomness and input, then re-computes the expected Yao's protocol message.

1.3 Technical Overview

Starting point. Our main CIR protocol securely evaluates a function of the form $(x, y) \mapsto$ [if $\Pi(x, y)$ then y else $\perp$]. Our starting point is Yao's textbook 2PC protocol, based on garbled circuits. Bob, with input y, garbles the circuit $\Pi(\cdot, y)$. Alice uses her input x as the choice bits in oblivious transfer (OT) to obtain a garbled input, on which she can evaluate the garbled circuit. As mentioned above, this protocol is not secure against an actively corrupt sender (Bob).

Conditionally checking the garbled circuit. Our key insight is that it suffices to check the correctness of the garbled circuit *only in the case that* $\Pi(x, y) = 1$. Suppose that whenever $\Pi(x, y) = 1$, the receiver learns the randomness that was used to garbled the circuit. In that case, the receiver can check the sender's garbled circuit for correctness, by re-garbling the circuit and comparing against what the sender sent. Opening the garbled circuit has the side-effect of invalidating its privacy guarantee, meaning that the sender's private input y is completely revealed. Usually this is undesirable, but in our case this leakage is fine, because the receiver is expected to learn y anyway when $\Pi(x, y) = 1$.

In more detail, our protocol works as follows:

- The sender runs Yao's protocol (for evaluating Π) normally, but with all of its randomness (for garbling and for the OT protocol) derived pseudorandomly from a single seed r.
- Π has a single output wire, whose garbled labels are W_0, W_1. Intuitively, the receiver learns W_1 *only* if the predicate Π evalutes to `true`. The sender sends an encryption of (y, r), using W_1 as the key.
- If $\Pi(x, y) = 0$, then W_1 remains hard to guess for the evaluator, so the encryption hides y and r, and the usual security of Yao's protocol applies, to completely hide y from the receiver.
- If $\Pi(x, y) = 1$, then the receiver learns W_1, which it can use to decrypt (y, r). Then it can then re-execute the sender's prescribed protocol behavior, which is a deterministic function of y and r. If it is discovered that the sender did not run the protocol correctly, the receiver will output $\perp$; otherwise, she will accept y as the protocol output.

Hence, whenever $\Pi(x, y) = 0$, the garbled circuit is not checked. To understand why this approach works, observe that the *only way* a corrupt sender can cause an honest receiver to output something other than $\perp$ is to honestly follow the specification of Yao's protocol.

Subtle issues. The preceding summary captures the essence of our approach, but glossses over a few subtle issues that require some care.

First, there is a circularity problem: The sender encrypts (y, r) using key W_1. But W_1 is a wire label in a circuit that was garbled using randomness r. The plaintext and key are correlated! We resolve the circularity by strategically introducing a random oracle: the sender uses $H(W_1)$ as the encryption key, where H is a random oracle.

Second, we send (y, r) not in an encryption but in a UC commitment. Then $H(W_1)$ is not the decryption key but the decommitment string. Importantly, this allows the simulator to extract (y, r). After extracting, the simulator has enough information to check whether the sender follows the protocol (the same check that the receiver would do, but the simulator can do it regardless of whether $\Pi(x, y) = 1$), and decide whether to deliver the receiver's output in the ideal world.

Finally, the receiver checks not only the garbled circuit, but also the sender's behavior in the oblivious transfers (OTs). Here it is crucial that we model OT as a *protocol,* whose random coins can be revealed, rather than as an *ideal functionality.* This introduces some subtlety into the analysis, as it breaks the usual abstractions of the UC model. We formally prove that everything works as expected, when the OT protocol is a 2-message, UC-secure OT protocol in the random oracle model.[1]

Fuzzy PAKE. To promote the CIR protocol to a PAKE protocol, we simply need to incorporate key agreement. In the case that $\Pi(x, y) = 1$ and the receiver does not detect cheating, both the sender and receiver will know the sender's randomness r for Yao's protocol. The two parties can therefore derive even more pseudorandomness from r, to use as their shared key. In all other cases, the receiver samples its output key uniformly, causing the agreement to fail.

The most important subtle issue is that fuzzy PAKE necessarily takes place over an *unauthenticated* channel, whereas typical UC-secure OT and commitment protocols are designed with authenticated channels in mind. In other words, UC security is proven in a setting where, if both parties are honest, the adversary cannot tamper with their protocol messages. Contrast that situation with that of (fuzzy) PAKE, whose ideal functionality explicitly considers an active person-in-the-middle attack on the communication between two honest parties.

Previous fuzzy PAKE protocols [9, 6] handle this issue by appealing to the "split functionality" technique of Barak et al. [1]. This is a general-purpose compiler that allows one to design an MPC protocol for authenticated channels but execute it on an unauthenticated channel. This approach would work here, but it involves adding digital signatures to every protocol message, and adding two extra rounds to initialize and agree on signing keys. We take a different approach and directly prove security of our fuzzy PAKE protocol, by assuming that the underlying UC OT and commitment protocols satisfy a natural non-malleability property. In this way, our final fuzzy PAKE protocol has only 2 message flows.

Information leakage in fuzzy PAKE. Note that in the event that the passwords are close, the receiver also learns the sender's password pw. We think this downside is acceptable for two reasons. First, if pw' is so similar to pw that we are willing to authenticate anyone who knows pw', then it does minimal harm to further reveal pw to them. Second, suppose we are running some fuzzy PAKE protocol that does not leak pw explicitly, but does reveal if $\Pi(\mathsf{pw}, \mathsf{pw}') = 1$. (i.e, if pw and pw' are close). As soon as an adversary finds some "close-enough" password guess, it can perform a binary search and identify pw with a number of guesses logarithmic in the number of points close to pw'. Thus, the leakage of pw is somewhat inevitable as long as PAKE is run repeatedly with the same pw and the adversary knows some close-enough pw'.

[1] Interestingly, we are not sure whether a UC-secure OT protocol with more than 2 rounds would necessarily work in our construction.

Explicit Authentication Our protocol comes with an additional advantage: because the receiver knows if authentication succeeds or not, one-sided explicit authentication can be achieved for free. Explicit authentication for the sender can be achieved with one extra key-confirmation message.

2 Preliminaries

2.1 Garbling

We present definitions for garbled circuits (garbling schemes), which are specialized to the case of circuits with a single output wire.

Definition 2.1. *Let $\mathcal{C}$ be a class of functions, where each function has a single output wire. A* ***garbling scheme*** *for $\mathcal{C}$ consists of the following algorithms:*

- $\mathsf{Gb}(1^\lambda, f) \to (F, e, d^*)$*: A randomized algorithm that takes a circuit $f \in \mathcal{C}$ as input, and outputs garbled circuit F, encoding information e, and decoding information d^*. (We deviate from typical definitions for garbling by letting d^* denote the "true" output label for the single output wire.)*

  We generally omit the 1^λ argument to Gb *from our notation; the security parameter is implicitly understood.*

- $\mathsf{En}(e, x) = (e[1, x[1]], e[2, x[2]], \ldots)$*:We insist that the garbling scheme is* ***projective*** *(in the terminology of [3]), meaning that the encoding information e is an $|x| \times 2$ array of wire labels, and the garbled encoding of input x is given as $\mathsf{En}(e, x)$ defined above.*

- $\mathsf{Ev}(F, X) \to d'$*: A deterministic algorithm that takes a garbled circuit F and garbled input encoding X as input, and outputs a* single wire label *d'.*

We further require the following properties to hold:

- ***(One-sided) perfect correctness:*** *For every $f \in \mathcal{C}$ and every x, if $f(x) = 1$ then the following process outputs* `true` *with probability 1:*

$$\boxed{\begin{array}{l}(F, e, d^*) \leftarrow \mathsf{Gb}(f) \\ \textit{output } \mathsf{Ev}\big(F, \mathsf{En}(e, x)\big) = d^*\end{array}}$$

- ***Security:*** *We encapsulate several security requirements into a single definition. We demand that there exists a simulator* Sim *such that, for all $f \in \mathcal{C}$ and all x such that $f(x) = 0$, the following two distributions are indistinguishable:*

$$\boxed{\begin{array}{l}(F, e, d^*) \leftarrow \mathsf{Gb}(f) \\ \textit{output } \big(F, \mathsf{En}(e, x), d^*\big)\end{array}} \approx \boxed{\begin{array}{l}(F, X) \leftarrow \mathsf{Sim}() \\ d^* \leftarrow \{0,1\}^\lambda \\ \textit{output } (F, X, d^*)\end{array}}$$

  The definition expresses two things: First, when $f(x) = 0$, the garbled circuit + garbled input leak nothing. In particular, they hide the choice of circuit $f \in \mathcal{C}$. (One may also want to hide this in the case that $f(x) = 1$, but this is not necessary in our protocol.) Second, when $f(x) = 0$, given F and X it is hard to guess the `true` *output wire label d^*, because it is pseudorandom. This is a strong (real vs random) definition of the standard* authenticity *property for garbling schemes.*

Suggested instantiations: Any standard garbling scheme (e.g., [33, 29]) will provide the necessary properties, although our requirements are not written in the standard way. We briefly explain how to translate the concepts to our notation.

Let f be a function and define $\mathcal{C} = \{f(\cdot, y) \mid y \in \mathcal{Y}\}$. In our notation, garbling a function $f(\cdot, y) \in \mathcal{C}$ is analogous to garbling the function f and including a garbled encoding of input y. The standard garbling property of *obliviousness* ensures that the garbled input hides y; this corresponds to hiding the choice of function from $\mathcal{C}$ in our notation. The function f has a single output wire and in our notation we let d^* denote the `true` label on that wire. The standard garbling property of *authenticity* ensures that it is hard to guess d^* given a garbled input that evaluates to `false`.

2.2 2-message [1-sided-]UC-secure Oblivious Transfer

Yao's protocol requires oblivious transfer (OT). We eventually prove UC security of our protocol, and the reader might expect to see our protocol written in terms of an ideal UC functionality. However, our protocol breaks the abstraction of a ideal functionality! We must write our protocol in terms of an OT *protocol,* because the sender sometimes reveals its protocol randomness to the receiver, who then re-executes the protocol to check for correctness. Thus, in this section we define OT in terms of concrete protocols.

Definition 2.2. *A **2-message OT protocol** (in the random oracle model) consists of the following algorithms:*

- $\mathsf{OT}_1^H(b) \to (M_1, \sigma_1)$*: A randomized algorithm that on input a choice bit b, produces the receiver's protocol message M_1 and internal state σ_1.*
- $\mathsf{OT}_2^H(v_0, v_1, M_1) \to M_2$*: A randomized algorithm that on input the sender's values (v_0, v_1) and a first protocol message M_1, produces the sender's protocol message M_2.*
- $\mathsf{OT}_3^H(\sigma_1, M_2) \to v'$*: A deterministic algorithm that when given the receiver's internal state σ_1 and the sender's protocol message M_2, produces the receiver's output v'.*

*The protocol is **perfectly correct** if, whenever the 3 algorithms are executed as shown above, $v' = v_b$. Formally, for all b, v_0, v_1, the following process outputs `true` with probability 1:*

$$\boxed{\begin{array}{l} H \leftarrow \textit{random oracle} \\ (M_1, \sigma_1) \leftarrow \mathsf{OT}_1^H(b) \\ M_2 \leftarrow \mathsf{OT}_2^H(v_0, v_1, M_1) \\ \textit{output } \mathsf{OT}_3^H(\sigma_1, M_2) = v_b \end{array}}$$

We require certain security properties for such an OT protocol. Below is the definition of UC security against a corrupt receiver, but specialized for the case of 2-message protocols in the random oracle model:

Definition 2.3. *A 2-message OT protocol is **UC-secure against a corrupt receiver** if there exist an efficient simulator $\mathsf{Sim}_R^{\mathsf{OT}} = (\mathsf{Sim}_{R,H}^{\mathsf{OT}}, \mathsf{Sim}_{R,1}^{\mathsf{OT}}, \mathsf{Sim}_{R,2}^{\mathsf{OT}})$ such that for all efficient adversaries, the following two interactions are indistinguishable:*

$$\boxed{\begin{array}{l} H \leftarrow \textit{random oracle} \\ \textit{give adversary oracle access to } H \\ \textit{receive } (M_1, v_0, v_1) \textit{ from adversary} \\ M_2 \leftarrow \mathsf{OT}_2^H(v_0, v_1, M_1) \\ \textit{give } M_2 \textit{ to adversary} \end{array}} \approx \boxed{\begin{array}{l} \textit{give adversary oracle access to } \mathsf{Sim}_{R,H}^{\mathsf{OT}} \\ \textit{receive } (M_1, v_0, v_1) \textit{ from adversary} \\ b := \mathsf{Sim}_{R,1}^{\mathsf{OT}}(M_1) \\ M_2 \leftarrow \mathsf{Sim}_{R,2}^{\mathsf{OT}}(v_b) \\ \textit{give } M_2 \textit{ to adversary} \end{array}}$$

We assume that all 3 of the $\mathsf{Sim}_{R,\star}^{\mathsf{OT}}$ algorithms share common state, which is not expressed explicitly as part of the interaction's pseudocode — i.e., the simulator can use what it learned playing the role of the

random oracle while extracting the adversary's input. Intuitively, the simulator plays the role of the random oracle ($\mathsf{Sim}^{\mathsf{OT}}_{R,H}$), can extract the corrupt receiver's choice bit ($\mathsf{Sim}^{\mathsf{OT}}_{R,1}$) from M_1, and can simulate the honest sender's protocol message ($\mathsf{Sim}^{\mathsf{OT}}_{R,2}$) using only its ideal OT output v_b.

We do not require full-fledged UC security for a corrupt sender, only a mild indistinguishability requirement:

Definition 2.4. *A 2-message OT protocol has **indistinguishable first messages** if there exists an efficient simulator $\mathsf{Sim}^{\mathsf{OT}}_S = (\mathsf{Sim}^{\mathsf{OT}}_{S,H}, \mathsf{Sim}^{\mathsf{OT}}_{S,1})$ such that for all efficient adversaries, the following two interactions are indistinguishable:*

$H \leftarrow$ *random oracle* *give adversary oracle access to* H *receive* b *from adversary* $(M_1, -) \leftarrow \mathsf{OT}_1^H(b)$ *give* M_1 *to adversary*	$\approx$	*give adversary oracle access to* $\mathsf{Sim}^{\mathsf{OT}}_{S,H}$ *receive* b *from adversary* $M_1 \leftarrow \mathsf{Sim}^{\mathsf{OT}}_{S,1}()$ *give* M_1 *to adversary*

As above, the $\mathsf{Sim}^{\mathsf{OT}}_{S,\star}$ algorithms may share common state.

Proposition 2.5. *If a 2-message OT protocol is UC-secure (under the standard UC security definition [7, Section 4.1]), then it also satisfies Definition 2.3 and 2.4.*

Theorem 2.3 follows directly from the UC security definition. More specifically, consider the ideal world of the UC-security game with a corrupt receiver (with a dummy real-world adversary). In this setting, the simulator receives a message from the environment (M_1) to be forwarded to the honest sender, and then must send a choice bit b to the OT functionality. It receives output from the OT functionality (v_b), then finally must send a simulated honest sender message (M_2) to the environment. This exactly matches the ideal setting in theorem 2.3, and UC-security states that M_2 is indistinguishable from the honest sender message in the real setting. Likewise, in the corrupt sender ideal-world game, the simulator must simulate an honest receiver message before receiving anything from the environment.

Suggested instantiations. Any 2-message UC-secure OT protocol will suffice. Examples include those of Peikert-Vaikuntanathan-Waters (PVW) [27], Masny-Rindal [22], and McQuoid-Rosulek-Roy [23].

In situations where many OT instances are required (significantly more than the security parameter), OT extension [2] is generally used. However, OT extension (of chosen-input OT) is not possible in 2 rounds [13], so it is not clear how to make our approach compatible with OT extension. In this way, our protocol is disadvantaged for circuits with very large inputs (thus a large number of OTs needed), compared to plain Yao's protocol.

2.3 Non-interactive [1-sided-]UC-secure Commitment

Our protocol also requires a non-interactive commitment protocol.

Definition 2.6. *A **non-interactive commitment protocol** in the random oracle model consists of the following algorithms:*

- $\mathsf{Com}^H(m; r) \to C$*: On input a message m and randomness r, outputs a commitment C.*
- $\mathsf{Decom}^H(C, r) \to m/\perp$*: Given a commitment C and its randomness r, produces an output m (or error indicator $\perp$).*

*The protocol is **perfectly correct** if for all m, the following process always outputs* `true`*:*

$$\boxed{\begin{array}{l} H \leftarrow \textit{random oracle} \\ r \leftarrow \{0,1\}^\lambda \\ C := \mathsf{Com}^H(m;r) \\ \textit{output } \mathsf{Decom}^H(C,r) == m \end{array}}$$

Our commitment syntax is slightly non-standard, since Decom does not require the payload (m) of the commitment, only its randomness (r). Instead, we want the receiver to *learn* m as a result of running Decom.

We require full UC security for a corrupt sender — i.e., a simulator should be able to extract the value of an adversarially generated commitment. Specialized to our non-interactive structure, the definition is as follows:

Definition 2.7. *A non-interactive commitment scheme is* ***UC-secure against a corrupt sender*** *if there exists an efficient simulator* $\mathsf{Sim}_S^{\mathsf{Com}} = (\mathsf{Sim}_{S,H}^{\mathsf{Com}}, \mathsf{Sim}_{S,1}^{\mathsf{Com}})$ *such that for all efficient adversaries, the following interactions are indistinguishable:*

$$\boxed{\begin{array}{l} H \leftarrow \textit{random oracle} \\ \textit{give adversary oracle access to } H \\ \textit{receive } (C,r) \textit{ from adversary} \\ \textit{output } \mathsf{Decom}^H(C,r) \end{array}} \approx \boxed{\begin{array}{l} \textit{give adversary oracle access to } \mathsf{Sim}_{S,H}^{\mathsf{Com}} \\ \textit{receive } (C,r) \textit{ from adversary} \\ m^* := \mathsf{Sim}_{S,1}^{\mathsf{Com}}(C) \\ \textit{if } \mathsf{Decom}^{\tilde{H}}(C,r) \neq \bot \textit{: output } m^* \\ \textit{else: output } \bot \end{array}}$$

As above, the $\mathsf{Sim}_{S,\star}^{\mathsf{Com}}$ *algorithms may share common state.*

The meaning of the security definition is the following: If the adversary produces a commitment C *and a value* r *such that* $\mathsf{Decom}(C,r) \neq \bot$*, then the simulator (*$\mathsf{Sim}_{S,1}^{\mathsf{Com}}$*) is able to extract the result of* $\mathsf{Decom}(C,r)$ *from* only C*. In particular, it cannot be possible that* $\mathsf{Decom}(C,r)$ *and* $\mathsf{Decom}(C,r')$ *are different and both not* $\bot$*.*

We do not require full-fledged UC security for the case of a corrupt receiver. Instead, we require only the following simple (and standard) hiding property:

Definition 2.8. *A non-interactive commitment scheme is* ***hiding*** *if for all efficient adversaries, the following games are indistinguishable:*

$$\boxed{\begin{array}{l} H \leftarrow \textit{random oracle} \\ \textit{give adversary oracle access to } H \\ \textit{receive } m \textit{ from adversary} \\ (C,-) \leftarrow \mathsf{Com}^H(m) \\ \textit{give } C \textit{ to adversary} \end{array}} \approx \boxed{\begin{array}{l} H \leftarrow \textit{random oracle} \\ \textit{give adversary oracle access to } H \\ \textit{receive } m \textit{ from adversary} \\ (C,-) \leftarrow \mathsf{Com}^H(0^{|m|}) \\ \textit{give } C \textit{ to adversary} \end{array}}$$

Suggested instantiations. As mentioned above, our commitment syntax is slightly non-standard, with the receiver only learning the payload m as a result of running Decom. We can achieve this feature by augmenting textbook random-oracle commitments in the following simple way:

- To commit to message m using randomness r, compute $k_0 \| k_1 = H(r)$, where $|k_0| = 2\lambda$ and $|k_1| = \lambda$. Output $C = (k_0, \mathsf{Enc}(k_1, m))$, where Enc is an authenticated symmetric encryption scheme.
- Upon learning r, the receiver can compute $H(r)$ and verify that it begins with k_0. If so, it decrypts the ciphertext using k_1 (the last λ bits of $H(r)$).
- The simulator can easily extract by identifying the (with overwhelming probability) unique oracle query of the form $r \mapsto k_0 \| \cdot$, from which it can obtain k_1 and decrypt the ciphertext to extract m. Importantly, we do not require any equivocation property, and thus it suffices to use a standard encryption scheme to encode m.

3 Conditionally Input-Revealing 2PC

First, we present a UC-secure protocol for conditionally input-revealing (CIR) functions. We formalize the goal via the CIR functionality in Figure 1. Informally, the CIR functionality gives the sender's input y to the receiver if $\Pi(x, y) = 1$ (where x is the receiver's input), and otherwise gives output $\perp$.

Parameters:
- input spaces $X = \{0,1\}^n$ and Y, where Y contains a special element $\perp$.
- predicate Π, such that $\Pi(x, \perp) = 0$ for all $x \in X$.

Behavior:
await input $x \in X$ from the receiver and $y \in Y$ from the sender
if $\Pi(x, y) = 1$, output y to the receiver; otherwise output $\perp$ to the receiver

Figure 1: Ideal functionality $\mathcal{F}_{\text{CIR}}$ for conditionally input-revealing 2PC.

3.1 Protocol Description

Our protocol is presented formally in Figure 2. It uses the following building blocks:

- $\mathsf{OT}_{\{1,2,3\}}$: A 2-message OT protocol (Section 2.2).
- $(\mathsf{Com}, \mathsf{Decom})$: A non-interactive commitment protocol (Section 2.3).
- $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev})$: A garbling scheme (Section 2.1).
- $H_{\mathsf{OT}}, H_{\mathsf{Com}}, H^*$: Random oracles for the OT, commitment, and CIR protocols, respectively.

The protocol follows the outline described in Section 1.3. It is parameterized by a predicate $\Pi(x, y)$, and we write $\Pi_y(x) = \Pi(x, y)$ — thus, Π_y is simply Π with y hard-coded. Essentially, the parties run Yao's protocol where the sender (who holds y) garbles the circuit Π_y. The sender derives its randomness for garbling and for OT from a pseudorandom seed r. Finally, the sender commits to (y, r), using $H^*(d^*)$ as the randomness, where d^* is the `true` garbled output label.

The receiver runs Yao's protocol as usual, eventually evaluating the garbled circuit to obtain garbled output (label) d'. The receiver uses d' in an attempt to decommit to the sender's commitment. If successful, the receiver learns (y, r) and can check whether the sender behaved correctly in the garbling, OTs, and commitment steps. If so, the receiver will output y; otherwise, they output $\perp$.

We write the sender's protocol behavior as a subroutine PROT, since the receiver also needs to perform and verify the same steps.

3.2 Lemma For 2-Message OT

Our security proof is delicate, and crucially relies on the fact that OT is realized by a 2-message protocol. The reason is the following observation about the simulator's behavior for a corrupt receiver: The simulator always needs to extract the corrupt receiver's input from its first OT message. If the ideal functionality outputs $\perp$, the simulator must *simulate* the sender's OT response. But if the ideal functionality outputs $y \neq \perp$, the simulator must *honestly generate* the sender's OT response, because in this case we expect the receiver to learn the sender's randomness. In other words, the simulator needs to be able to extract, before it even decides whether the sender is simulated or honest. Below, we show that this is possible in a 2-message OT protocol, because extraction happens before the sender speaks.

Lemma 3.1. *Suppose* OT *is a two-message OT protocol that is UC-secure against a corrupt receiver (Definition 2.3), with simulator* $\mathsf{Sim}^{\mathsf{OT}}_{R,\star}$*. Then for any efficient adversary, the following games are indistinguishable:*

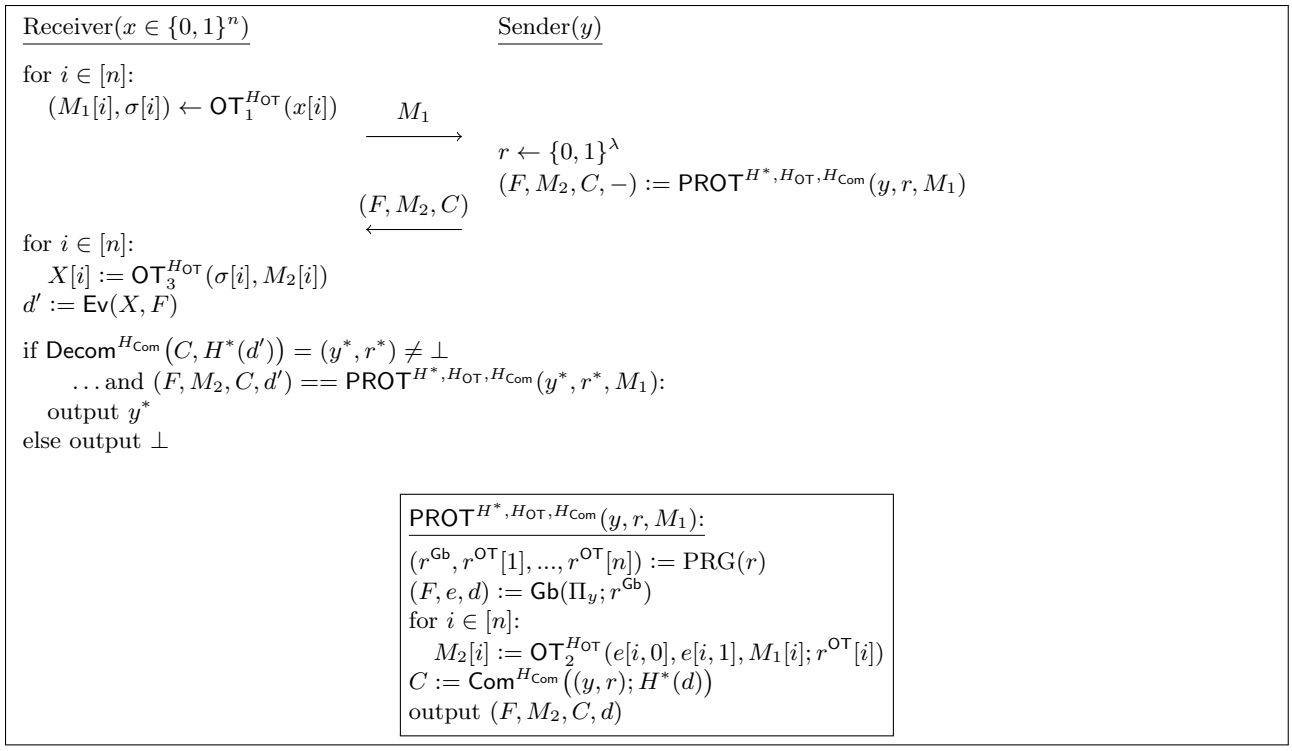

Figure 2: Our CIR-2PC protocol. Each ith OT instance receives a distinct SID $sid||i$, where sid is the CIR session ID.

give oracle access to $\mathsf{Sim}_{R,H}^{\mathsf{OT}}$ *receive* (M_1, v_0, v_1) *from adversary* $b := \mathsf{Sim}_{R,1}^{\mathsf{OT}}(M_1)$ $M_2 \leftarrow \mathsf{OT}_2^{\mathsf{Sim}_{R,H}^{\mathsf{OT}}}(v_0, v_1, M_1)$ *give* M_2 *to adversary*	*give oracle access to* $\mathsf{Sim}_{R,H}^{\mathsf{OT}}$ *receive* (M_1, v_0, v_1) *from adversary* $b := \mathsf{Sim}_{R,1}^{\mathsf{OT}}(M_1)$ $M_2 \leftarrow \mathsf{Sim}_{R,2}^{\mathsf{OT}}(v_b)$ *give* M_2 *to adversary*

Note: the second game is exactly the "ideal" interaction from Definition 2.3. In the first interaction, the simulator extracts the receiver's input b but does not use the result; instead, it generates M_2 honestly.

Proof. Compare the first interaction to the "real" interaction from Definition 2.3. Their only difference is in the oracle that is provided to the adversary and to OT_2. (Our interaction also calls $\mathsf{Sim}_{R,1}^{\mathsf{OT}}$, but its result is never used.) Definition 2.3 obviously implies that oracles H_{OT} and $\mathsf{Sim}_{R,H}^{\mathsf{OT}}$ are indistinguishable. Thus, our first interaction is indistinguishable from the "real" interaction in Definition 2.3. Then it is also indistinguishable from the "ideal" interaction in that definition, which is the second interaction in this lemma. □

3.3 Security proof

Theorem 3.2. *The protocol in Figure 2 UC-realizes $\mathcal{F}_{CIR}$ (Figure 1), assuming:*

- $\mathsf{OT}_{1,2,3}$ *is a 2-message 1-out-of-2 OT scheme that is UC-secure in the ROM (Section 2.2),*
- $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev})$ *is a garbling scheme with the properties defined in section 2.1 with respect to circuit class* $\mathcal{C} = \{\Pi_y \mid y \in Y\}$*,*

- $(\mathsf{Com}, \mathsf{Decom})$ *is a non-interactive 1-round commitment scheme that is UC-secure in the ROM (Section 2.3).*

We prove the theorem by separately considering the case of corrupted sender and corrupted receiver.

3.3.1 Corrupt sender

The simulator for a corrupt sender is shown in fig. 3. We assume the adversary is a dummy adversary and that the simulator communicates directly with the environment. Essentially, the simulator generates simulated OT messages on behalf of the honest receiver. There are n instances of OT, and $\mathsf{Sim}^{\mathsf{OT}}_{S,1,i}$ denotes the ith independent instance of the OT simulator. (Each instance also has its own random oracle.) The simulator can extract the contents of the adversary's commitment (y^*, r^*), and run the PROT subroutine to check that the adversary's protocol messages. If these are consistent with respect to PROT, the simulator sends y^* to the ideal functionality; otherwise, it sends $\perp$.

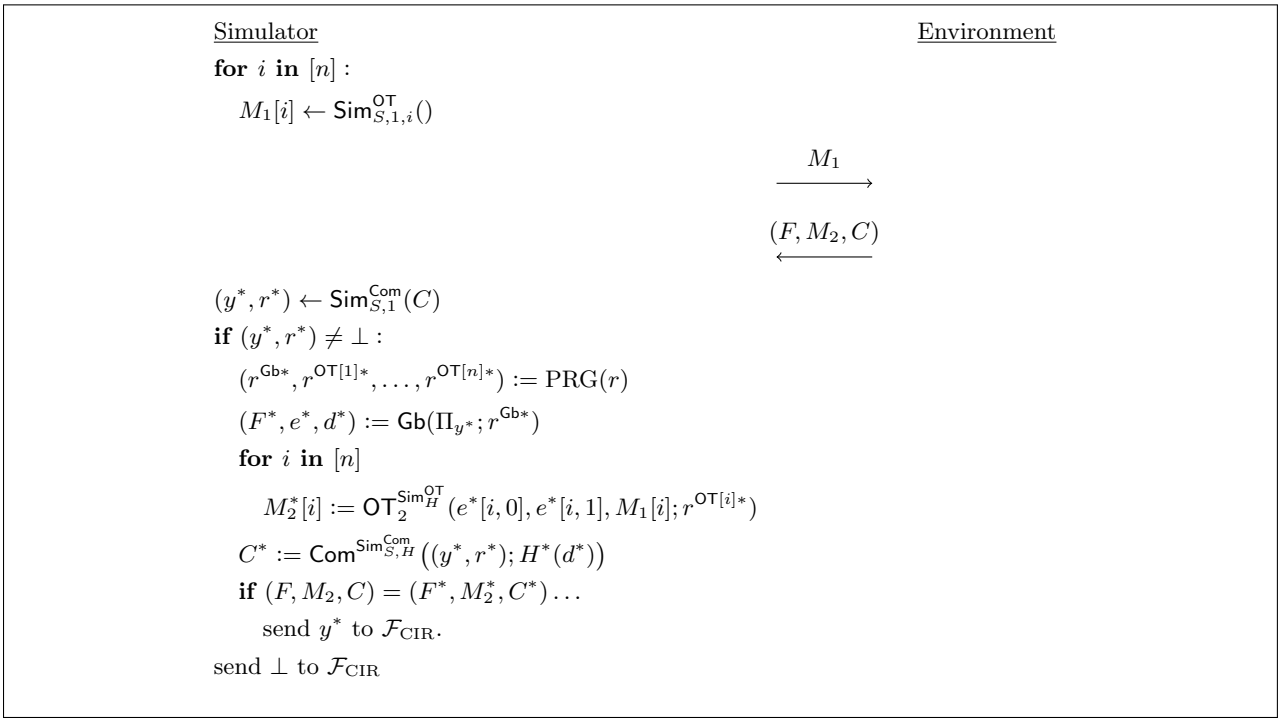

Figure 3: Simulator for a corrupt sender. The simulator responds to H_{OT} and H_{Com} random oracle queries using simulator routines $\mathsf{Sim}^{\mathsf{OT}}_{S,H,\cdot}$ and $\mathsf{Sim}^{\mathsf{Com}}_{S,H}$. It simulates oracle H^* honestly.

We prove that the simulation is indistinguishable from the real world through the following sequence of hybrids. All hybrids are formally described in Appendix C.1.

$\textsc{hybrid}_0$(fig. 10): In this hybrid, we imagine an ideal world with a variant of $\mathcal{F}_{\mathrm{CIR}}$ that leaks the honest receiver's input x to the simulator. The simulator then runs the honest receiver's protocol honestly with the adversary, and reports the receiver's output to the environment. As such, the hybrid is identical to the real world.

$\textsc{hybrid}_1$(fig. 11): In the previous hybrid, the simulator computes OT output X, evaluates the garbled circuit on X to learn d', uses d' to open the commitment C, and finally re-computes the sender's "expected" protocol messages using PROT. In this hybrid, the simulator directly extracts the contents of commitment C (as y^*, r^*) and then performs the PROT consistency check. If the check succeeds, and if $\Pi(x, y^*) = 1$, then the simulator outputs y^* on behalf of the honest receiver. Otherwise, it outputs $\perp$. (In order to perform

extraction, the simulator also replaces H_{Com} with the simulated oracle $\mathsf{Sim}^{\mathsf{Com}}_{S,H}$.) We argue that these hybrids are indistinguishable:

Case 1: HYBRID_1 outputs $y^* \neq \perp$. This only happens if the adversary behaved honestly according to input y^* and randomness r, and if $\Pi(x, y^*) = 1$. In this case, we can apply the perfect correctness of the OT protocol, garbling scheme, and commitment, to conclude that in HYBRID_0 the simulator would have evaluated a correctly garbled circuit Π_y on correct garbled input x, and obtained the correct output label d^*. Then it would have succeeded in opening the correctly-generated commitment and run the PROT consistency check and eventually output y^* as well.

Case 2: HYBRID_1 outputs $\perp$. If the simulator in HYBRID_0 would have failed to open C, it would have also output $\perp$ and we are done. Otherwise, assume the simulator in HYBRID_0 would have successfully opened C. Then by the UC-security of the commitment scheme (Definition 2.7), the result of decommitment must match the (y^*, r^*) extracted by the simulator in HYBRID_1. This means that both hybrids perform the PROT consistency check using the same input. If the consistency check fails (the sender did not follow the protocol), then both hybrids output $\perp$, and we are done. Otherwise, HYBRID_1 must have output $\perp$ because $\Pi(x, y^*) = 0$. Since the sender followed the protocol, we can apply the correctness of OT, garbling, and commitment (as in the previous case) to argue that the simulator in HYBRID_0 must have obtained garbled input $d' \neq d^*$. But this would be a contradiction since the receiver outputs $\perp$ if $d' \neq d^*$. So overall, the hybrids give matching output.

HYBRID_2(fig. 12): Same as the previous hybrid, except that it no longer checks that $d' = d^*$ before outputting y^*. Assume that $(y^*, r^*) \neq \perp$, $\Pi(x, y^*) = 1$, and $(F, M_2, C) = (F^*, M_2^*, C^*)$. Because $(F, M_2) = (F^*, M_2^*)$, X is the proper garbled encoding for F, and F is a proper garbling of Π_{y^*} with d^* as the wire label encoding true. So, perfect correctness of the garbling scheme implies that $d' = d^*$. So, checking $d' = d^*$ is redundant. In all other cases, this check does not matter, so it can be removed.

HYBRID_3(fig. 13): Same as the previous hybrid, except that instead of checking $\Pi(x, y^*)$ and delivering the honest receiver's output, it sends y^* or $\perp$ to $\mathcal{F}_{\text{CIR}}$, which does the same check and same output delivery. The hybrids are identical.

HYBRID_4(fig. 14): Same as the previous hybrid, except that instead of generating M_1 by running the protocol honestly, it generates them using $\mathsf{Sim}^{\mathsf{OT}}_1$. In doing so, it also responds to H_{OT} oracle queries using $\mathsf{Sim}^{\mathsf{OT}}_H$. The hybrids are indistinguishable by the security of the OT protocol.

Ideal world: This last hybrid receives the honest party's input x from the "leaky" functionality, but doesn't use it. Thus, the hybrid is indistinguishable from the ideal world involving the "normal" functionality that doesn't give x to the simulator. This completes the proof.

3.3.2 Corrupt Receiver

The simulator for this case is shown in fig. 4. The simulator first extracts the adversary's input from its OT choice bits, and sends the input to the ideal functionality. If it receives $y \neq \perp$ in response, then it behaves exactly as an honest sender on input y. Otherwise, it simulates the garbled circuit, simulates the OT protocol response, and gives a dummy commitment.

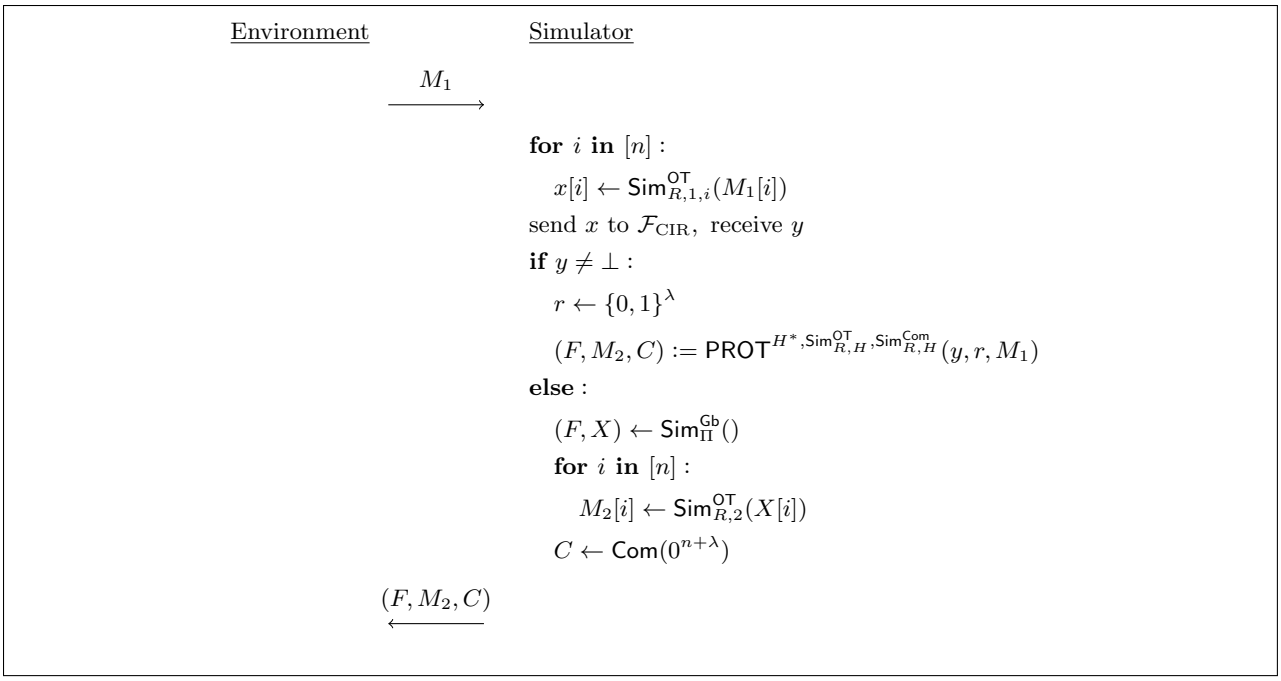

Figure 4: Simulator for a corrupt receiver. The simulator responds to H_{OT} and H_{Com} oracle queries using $\mathsf{Sim}^{\mathsf{OT}}_{R,H,\cdot}$ and $\mathsf{Sim}^{\mathsf{Com}}_{R,H}$, respectively. It simulates H^* honestly.

Again, all hybrids are formally described in Appendix C.2.

HYBRID$_0$(fig. 15): In this hybrid, we imagine an ideal world with a variant of $\mathcal{F}_{\mathrm{CIR}}$ that immediately leaks the honest sender's input y to the simulator. It then runs the protocol honestly on the sender's input. As such, the hybrid is identical to the real world.

HYBRID$_1$(fig. 16): Same as the previous hybrid, except that random oracle(s) H_{OT} are replaced with simulated oracles $\mathsf{Sim}^{\mathsf{OT}}_{R,H,\cdot}$. The change is indistinguishable by the security of the OT protocol. Additionally, we may let the simulator extract (using $\mathsf{Sim}^{\mathsf{OT}}_{R,1,i}$) the adversary's OT inputs. However, the simulator does nothing with these values (yet), so this has no effect on the adversary's view.

HYBRID$_2$(fig. 17): Same as the previous hybrid, except in this hybrid the simulator uses the input x^* that it extracted from OT. It sends x^* to the leaky $\mathcal{F}_{\mathrm{CIR}}$ and learns the receiver's ideal output y' (either y or $\perp$, depending on whether $\Pi(x^*, y) = 0$). Note: we distinguish y (the honest receiver's input, leaked by this variant of $\mathcal{F}_{\mathrm{CIR}}$) and y' (the corrupt sender's output, which may equal y or $\perp$). This is merely a syntactical change, because the simulator does nothing with this knowledge of y'. (In the figure, we introduce an if-statement based on whether $y' = \perp$, but the bodies of both branches are identical.)

HYBRID$_3$(fig. 18): Same as the previous hybrid, except now in the case that $y' = \perp$, the simulator generates C as a dummy commitment, instead of a commitment to (y, r). (In the figure, we inline the PROT subroutine so we can make this small internal change.)The simulator will also now abort if H^* is ever queried on input d. If not for this artificial abort, the change would be indistinguishable by the hiding property of the commitment scheme.

Let p_i be the probability of this abort in hybrid i. If p_3 is negligible, then hybrids 2 and 3 are indistinguishable. Throughout the next steps of the proof, we will establish that $p_3 \approx \cdots \approx p_7$, and that p_7 is negligible, and thus p_3 is negligible.

HYBRID$_4$(fig. 19): Same as the previous hybrid, except that in the case $y' = \perp$, the simulator does not sample protocol randomness pseudorandomly; rather, it uses true randomness. The hybrids are indistinguishable because in the case of $y' = \perp$, hybrid 3 samples randomness by expanding r with a secure PRG, and this is the *only* place r is used. (In previous hybrids, r appeared in the commitment, but in hybrid 3 this commitment becomes a dummy commitment.)

HYBRID$_5$(fig. 20): Same as the previous hybrid, except, in the case that $y' = \perp$, instead of generating M_2 by running the OT protocol, the simulator generates it via $\mathsf{Sim}^{\mathsf{OT}}_{R,2}$. Because hybrid 4 generated M_2 by running the OT protocol completely honestly (with true randomness), the change is indistinguishable by Lemma 3.1.

HYBRID$_6$(fig. 21): Note that in the case of $y' = \perp$, the previous hybrid does not use all of the garbled input information e. It uses only $e[i, x[i]]$, as input $\mathsf{Sim}^{\mathsf{OT}}_{R,2,i}$ — in other words, it needs only $X = \mathsf{En}(e, x)$. This hybrid is as the previous hybrid, except that in the case of $y' = \perp$, instead of generating F and X honestly, it simulates them with $\mathsf{Sim}^{\mathsf{Gb}}_{\Pi}$. The change is indistinguishable by the obliviousness of the garbling scheme.

HYBRID$_7$(fig. 22): Same as the previous hybrid, except that we remove the artificial abort. In hybrid 6, the value d was chosen uniformly at random, and *independent* of the adversary's view. Thus, the abort condition (the adversary queries H^* at d) happens with negligible probability and can be removed.

Ideal world: Hybrid 7 initially receives the honest party's input y from the "leaky" functionality, but does not use it. Thus, the hybrid is indistinguishable from the ideal world involving the "normal" functionality that doesn't give y to the simulator. This completes the proof.

4 Fuzzy PAKE

4.1 The Fuzzy PAKE Functionality

The ideal functionality for fuzzy PAKE ([9]) is shown in Figure 5. In the functionality we assume that all passwords have length n. As in the previous section, the functionality is also parameterized by a symmetric predicate Π indicating whether two length-n bitstrings are "close".

4.2 Additional Properties of OT and Garbling

Oblivious transfer. Because PAKE protocols operate without authenticated channels, the existence of a man-in-the-middle adversary is a fundamental part of the security definition. (In our CIR protocol, we assume secure channels between the two parties.) Indeed, this is the main complication of the PAKE setting vs. the CIR setting of the previous section. As such, we require some underlying primitives (namely, OT) to be secure in the presence of a man-in-the-middle.

The following definition considers an adversary attacking the OT protocol as a man-in-the-middle. An honest receiver creates the first protocol message M_1; the adversary may relay M_1 or any message of its choice to the sender, who responds with M_2. The adversary may relay M_2 or any other message to the receiver, who produces output. This attack scenario should be indistinguishable from one in which a collection of simulators do the following:

- The honest receiver's M_1 can be simulated given no information about the choice bit.
- If the adversary relays M_1 honestly, then M_2 can be simulated given *no* information about the sender's OT input. Otherwise, the simulator can extract a choice bit b' from the adversary and simulate M_2 given only $v_{b'}$ (if the sender's OT input is v_0, v_1).

- On input $(\mathsf{NewSession}, sid, P, P', \mathsf{pw}, \mathsf{role} \in \{\mathsf{Receiver}, \mathsf{Sender}\})$ from P, if this is the first NewSession message for sid, or this is the second NewSession message for sid and there is a record $\langle P', P, \star, \mathsf{role}' \rangle$ with $\mathsf{role}' \neq \mathsf{role}$, then store $\langle P, P', \mathsf{pw}, \mathsf{role} \rangle$ and mark it fresh, and send $(\mathsf{NewSession}, sid, P, P', \mathsf{role})$ to $\mathcal{A}^*$.
- On $(\mathsf{TestPwd}, sid, P, \mathsf{pw}^*)$ from $\mathcal{A}^*$, if there is a record $\langle P, P', \mathsf{pw}, \mathsf{role} \rangle$ marked fresh, then do:
  - If $\Pi(\mathsf{pw}^*, \mathsf{pw}) = 1$, then mark the record correctguess. If $\mathsf{role} = \mathsf{Sender}$, send pw to $\mathcal{A}^*$, otherwise send "correct guess" to $\mathcal{A}^*$.
  - If $\Pi(\mathsf{pw}^*, \mathsf{pw}) = 0$, then mark the record interrupted and send "wrong guess" to $\mathcal{A}^*$.
- On $(\mathsf{NewKey}, sid, P, K^* \in \{0,1\}^\lambda)$ from $\mathcal{A}^*$, if there is a record $\langle P, P', \mathsf{pw}, \star \rangle$, and this is the first NewKey message for sid and P, then output (sid, K) to P, where K is defined as follows:
  - If the record is correctguess, then set $K := K^*$.
  - If the record is fresh, a key (sid, K') has been output to P', at which time there was a record $\langle P', P, \mathsf{pw}' \rangle$ marked fresh such that $\Pi(\mathsf{pw}, \mathsf{pw}') = 1$, then set $K := K'$.
  - Otherwise sample $K \leftarrow \{0,1\}^\lambda$.

  Finally, mark the record completed.

Figure 5: UC ideal functionality for fuzzy PAKE $\mathcal{F}_{\mathrm{FPAKE}}^{\Pi}$, parameterized by a similarity function Π.

- If the adversary relays both M_1, M_2 honestly, then the honest receiver's output is consistent with an honest execution. Otherwise, the simulator can extract v_0', v_1' from the adversary such that the honest receiver (with choice bit b) outputs v_b'.

Definition 4.1. *A two-message OT protocol is* ***non-malleable*** *if there exists a collection of simulators* $\mathsf{Sim}_\star^{\mathsf{OT}}$ *(that share state) such that the following interactions are indistinguishable for any stateful adversary* $\mathcal{A}$*.*

$$
\boxed{\begin{array}{l} b \leftarrow \mathcal{A}^H() \\ M_1 \leftarrow \mathsf{OT}_1^{H_{\mathsf{OT}}}(b) \\ (v_0, v_1, M_1') \leftarrow \mathcal{A}^H(M_1) \\ M_2 \leftarrow \mathsf{OT}_2^{H_{\mathsf{OT}}}(v_0, v_1, M_1') \\ M_2' \leftarrow \mathcal{A}^{H_{\mathsf{OT}}}(M_2) \\ v \leftarrow \mathsf{OT}_3^{H_{\mathsf{OT}}}(\sigma, M_2') \\ \text{output } \mathcal{A}^{H_{\mathsf{OT}}}(v) \end{array}} \approx \boxed{\begin{array}{l} b \leftarrow \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}() \\ M_1 \leftarrow \mathsf{Sim}_{S,1}^{\mathsf{OT}}() \\ (v_0, v_1, M_1') \leftarrow \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(M_1) \\ \textbf{if } M_1 = M_1' \textit{:} \\ \quad M_2 \leftarrow \mathsf{Sim}_{R,2}^{\mathsf{OT}}() \\ \textbf{else} \textit{:} \\ \quad b' \leftarrow \mathsf{Sim}_{R,1}^{\mathsf{OT}}(M_1') \\ \quad M_2 \leftarrow \mathsf{Sim}_{R,2}^{\mathsf{OT}}(v_{b'}) \\ M_2' \leftarrow \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(M_2) \\ \textbf{if } (M_1, M_2) = (M_1', M_2') \textit{:} \\ \quad \text{output } \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(v_b) \\ \textbf{else} \textit{:} \\ \quad (v_0', v_1') \leftarrow \mathsf{Sim}_{S,2}^{\mathsf{OT}}(M_2') \\ \quad \text{output } \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(v_b') \end{array}}
$$

Recall that in the proof of our CIR protocol, we considered an "intermediate" world in which we use the simulator to extract from an adversary's M_1 protocol message, but generate the response M_2 honestly (not simulated). We require something similar in the PAKE setting:

Lemma 4.2. *Suppose* OT *is a non-malleable two-message random oracle OT protocol, with simulators*

$\mathsf{Sim}_\star^{\mathsf{OT}}$. *Then the following interactions are indistinguishable for any adversary* $\mathcal{A}$*:*

$$
\boxed{
\begin{array}{l}
b \leftarrow \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}() \\
M_1 \leftarrow \mathsf{Sim}_{S,1}^{\mathsf{OT}}() \\
(v_0, v_1, M_1') \leftarrow \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(M_1) \\
\boxed{\begin{array}{l}
\textbf{if } M_1 \neq M_1'\textit{:} \\
\quad b' \leftarrow \mathsf{Sim}_{R,1}^{\mathsf{OT}}(M_1') \\
M_2 \leftarrow \mathsf{OT}_2^{\mathsf{Sim}_H^{\mathsf{OT}}}(v_0, v_1, M_1')
\end{array}} \\
M_2' \leftarrow \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(M_2) \\
\textbf{if } (M_1, M_2) = (M_1', M_2')\textit{:} \\
\quad \text{output } \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(v_b) \\
\textbf{else}\textit{:} \\
\quad (v_0', v_1') \leftarrow \mathsf{Sim}_{S,2}^{\mathsf{OT}}(M_2') \\
\quad \text{output } \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(v_b')
\end{array}
}
\approx
\boxed{
\begin{array}{l}
b \leftarrow \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}() \\
M_1 \leftarrow \mathsf{Sim}_{S,1}^{\mathsf{OT}}() \\
(v_0, v_1, M_1') \leftarrow \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(M_1) \\
\boxed{\begin{array}{l}
\textbf{if } M_1 = M_1'\textit{:} \\
\quad M_2 \leftarrow \mathsf{Sim}_{R,2}^{\mathsf{OT}}() \\
\textbf{else}\textit{:} \\
\quad b' \leftarrow \mathsf{Sim}_{R,1}^{\mathsf{OT}}(M_1') \\
\quad M_2 \leftarrow \mathsf{Sim}_{R,2}^{\mathsf{OT}}(v_{b'})
\end{array}} \\
M_2' \leftarrow \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(M_2) \\
\textbf{if } (M_1, M_2) = (M_1', M_2')\textit{:} \\
\quad \text{output } \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(v_b) \\
\textbf{else}\textit{:} \\
\quad (v_0', v_1') \leftarrow \mathsf{Sim}_{S,2}^{\mathsf{OT}}(M_2') \\
\quad \text{output } \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(v_b')
\end{array}
}
$$

Note that the second game in this lemma is identical to the second game in theorem 4.1.

We also require the following. It is very similar to the previous lemma, except: (1) the reference point is the "real" game of theorem 4.1, not the "ideal" game, and (2) the honest sender's protocol message M_2 is generated honestly, but with randomness chosen by the advesary. This reflects the fact that in our protocol, M_2 is computed using externally supplied randomness (from a PRG) that might be revealed.

Lemma 4.3. *Suppose* OT *is a non-malleable two-message random oracle OT protocol, with simulators* $\mathsf{Sim}_\star^{\mathsf{OT}}$*. Then the following interactions are indistinguishable for any adversary* $\mathcal{A}$*:*

$$
\boxed{
\begin{array}{l}
b \leftarrow \mathcal{A}^H() \\
M_1 \leftarrow \mathsf{OT}_1^{H_{\mathsf{OT}}}(b) \\
(v_0, v_1, M_1', r) \leftarrow \mathcal{A}^H(M_1) \\
M_2 := \mathsf{OT}_2^{H_{\mathsf{OT}}}(v_0, v_1, M_1'; r) \\
M_2' \leftarrow \mathcal{A}^{H_{\mathsf{OT}}}(M_2) \\
v \leftarrow \mathsf{OT}_3^{H_{\mathsf{OT}}}(\sigma, M_2') \\
\text{output } \mathcal{A}^{H_{\mathsf{OT}}}(v)
\end{array}
}
\approx
\boxed{
\begin{array}{l}
b \leftarrow \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}() \\
M_1 \leftarrow \mathsf{Sim}_{S,1}^{\mathsf{OT}}() \\
(v_0, v_1, M_1', r) \leftarrow \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(M_1) \\
\textbf{if } M_1 \neq M_1'\textit{:} \\
\quad b' \leftarrow \mathsf{Sim}_{R,1}^{\mathsf{OT}}(M_1') \\
M_2 := \mathsf{OT}_2^{\mathsf{Sim}_H^{\mathsf{OT}}}(v_0, v_1, M_1'; r) \\
M_2' \leftarrow \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(M_2) \\
\textbf{if } (M_1, M_2) = (M_1', M_2')\textit{:} \\
\quad \text{output } \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(v_b) \\
\textbf{else}\textit{:} \\
\quad (v_0', v_1') \leftarrow \mathsf{Sim}_{S,2}^{\mathsf{OT}}(M_2') \\
\quad \text{output } \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(v_b')
\end{array}
}
$$

Proofs of these lemmas are deferred to Appendix A.

Suggested Instantiation. In Appendix B we show that a variant of the PVW OT protocol [27] satisfies theorem 4.1, under a Diffie-Hellman-type assumption.

Garbling. In our PAKE protocol, M_1 is a vector of n OT protocol messages, one for every bit of the password. We must account for the situation that a man-in-the-middle adversary may disrupt some of the n OT instances and not others. In that case, the adversary will learn garbled input for a *proper subset* of input wires in the garbled circuit. We need to argue that the garbled circuit reveals nothing in this case.

(For example, even if the password predicate is that passwords agree in at least $n-2$ positions, and the adversary learns $n-1$ garbled inputs for the correct password.)

Definition 4.4. *A garbling scheme* Gb *for class of circuits* $\mathcal{C}$ *has* ***obliviousness under partial inputs*** *if the following games are indistinguishable:*

$\underline{\text{Real}(f \in \mathcal{C}, v, x)}$
assert $v \neq 1^{\lvert x \rvert}$
$(F, e, d) \leftarrow \mathsf{Gb}(f)$
$X = \mathit{Encode}(e, x)$
for i in $[\lvert x \rvert]$:
if $v[i] = 0 : X[i] = \perp$
output F, X, d

$\approx$

$\underline{\text{Simulated}(f \in \mathcal{C}, v, x)}$
assert $v \neq 1^{\lvert x \rvert}$
$(F, X) \leftarrow \mathsf{Sim}^{\mathsf{Gb}}()$
$d \leftarrow \{0,1\}^{\lambda}$
for i in $[\lvert x \rvert]$:
if $v[i] = 0 : X[i] = \perp$
output F, X, d

In these games, the adversary provides a bit vector v *that selects whether the adversary wants to learn the garbled input on that wire. The adversary must select a proper subset of input wires, and the resulting view leaks nothing about* f *(similar to theorem 2.1), regardless of the input* x*.*

In a standard garbling scheme, the evaluator must have wire labels for all wires that affect the output wire label in order to compute the output wire label. So, theorem 4.4 will be true for standard garbling schemes as long as $\mathcal{C}$ only contains circuits in which all input wire labels are used to compute the output wire label, which will be true of any reasonable circuit.

4.3 Our Fuzzy PAKE Protocol

Our fuzzy PAKE protocol is shown in fig. 6. It is mostly similar to the CIR protocol, with the following changes:

- The commitment now uses $H^*(d||M_1)$ as randomness rather than $H^*(d)$.
- The PRG that is used to generate randomness for Yao's protocol now also outputs a key K.
- The sender outputs this value K as the session key. The receiver outputs a random key whenever the CIR protocol would output $\perp$, and outputs K otherwise.

We use the following conventions about the session id:

- All protocol messages include the *sid* of the session it is in, which is omitted for simplicity.
- All RO inputs include a *sid* to achieve domain extension, which is also omitted. (Equivalently, each *sid* has its own separate random oracle.)
- Each ith OT instance receives a distinct SID $sid||i$.

4.4 Security Proof

Theorem 4.5. *The protocol in fig. 6 UC-realizes* $\mathcal{F}_{FPAKE}^{\Pi}$ *if:*

- $\mathsf{OT}_{1,2,3}$ *is a non-malleable two-message OT protocol in the ROM (Theorem 4.1),*
- $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev})$ *is a projective garbling scheme for circuit class* $\mathcal{C} = \{\Pi_y \mid y \in \{0,1\}^n\}$*, that satisfies the properties of Theorem 2.1 and is oblivious under partial inputs (Theorem 4.4), and*
- $(\mathsf{Com}, \mathsf{Decom})$ *is a non-interactive commitment scheme that is UC-secure in the ROM (Section 2.3).*

Proof. As standard in UC, assume the real adversary is "dummy", i.e., it merely passes all messages between the environment and the protocol parties.

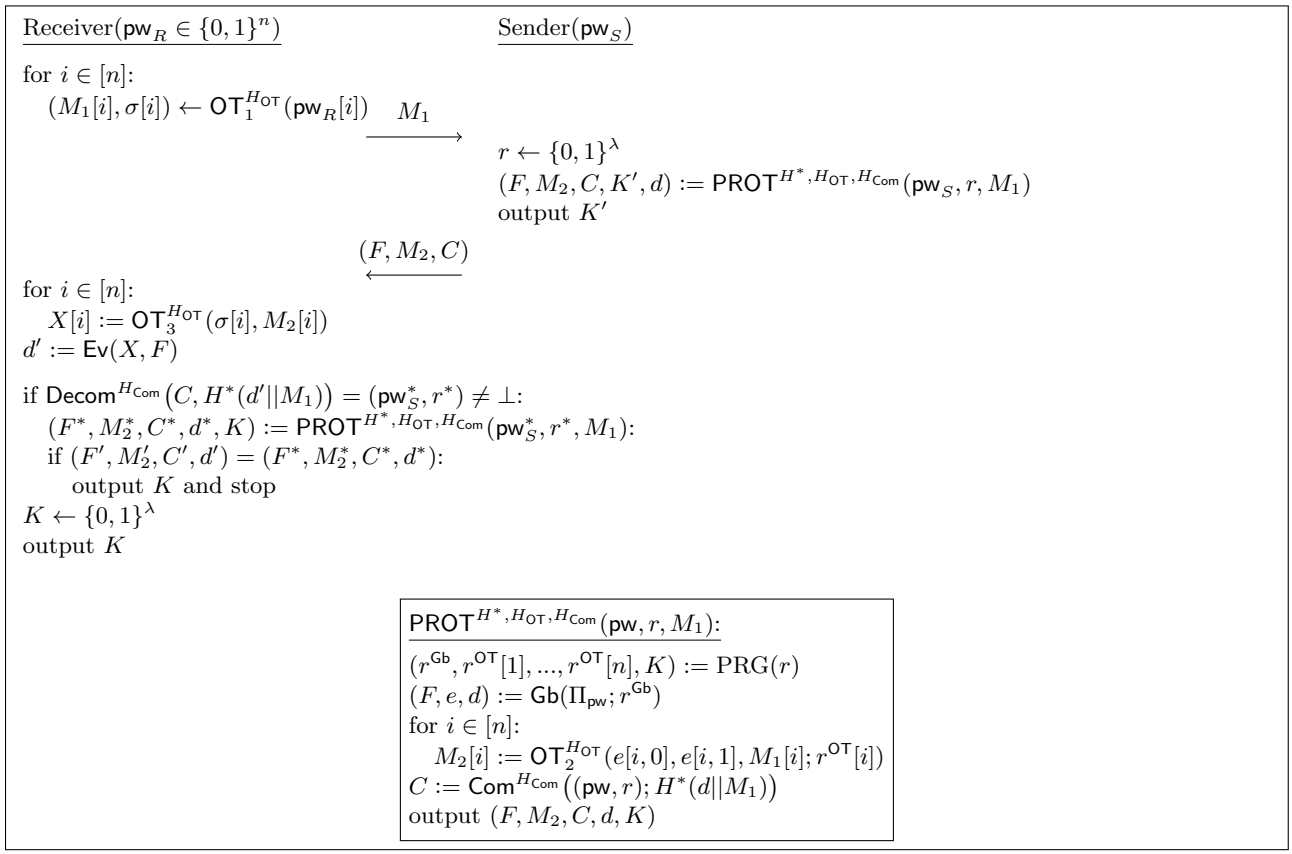

Figure 6: Our fuzzy PAKE protocol.

The simulator. The simulator is shown in Figure 7.

Hybrids. We now show that the simulator generates a view for the environment indistinguishable from the real view via a sequence of hybrids, starting from the real view and ending at the ideal view.

HYBRID_0(fig. 23): This is the real execution of the protocol. The sequence of events (after a session starts) is:

- Step 1: The adversary receives M_1 from the receiver, and sends M_1' to the sender.
- Step 2: The adversary and the environment receive (F, M_2, C) and K' from the sender, respectively. The adversary then sends (F', M_2', C') to the receiver.
- Step 3: The environment receives K from the receiver.

HYBRID_1(fig. 24): In HYBRID_1, the OT first messages $M_1[i]$ and oracle are simulated. The OT outputs $X[i]$ are also simulated, except when $(M_1[i], M_2[i]) = (M_1'[i], M_2'[i])$, in which case $X[i]$ is set to $e[i, \mathsf{pw}_R[i]]$. (We also let the OT simulator to generate a pw_R^*, although it is not currently used.) This change is indistinguishable from HYBRID_0 because:

- If $(M_1[i], M_2[i]) = (M_1'[i], M_2'[i])$, note that in HYBRID_0 the receiver's and the sender's inputs to the OTs are $\mathsf{pw}_R[i]$ and $\{e[i,0], e[i,1]\}$, respectively; therefore, correctness of the OT scheme implies the receiver will receive $e[\mathsf{pw}_R[i]]$[2], so there is no change.
- If $(M_1[i], M_2[i]) \neq (M_1'[i], M_2'[i])$, indistinguishability holds by theorem 4.3.

[2]An honest execution of the OT protocol uses true randomness, whereas here we use outputs of a PRG; however, since the OT correctness is perfect, this does not cause an issue.

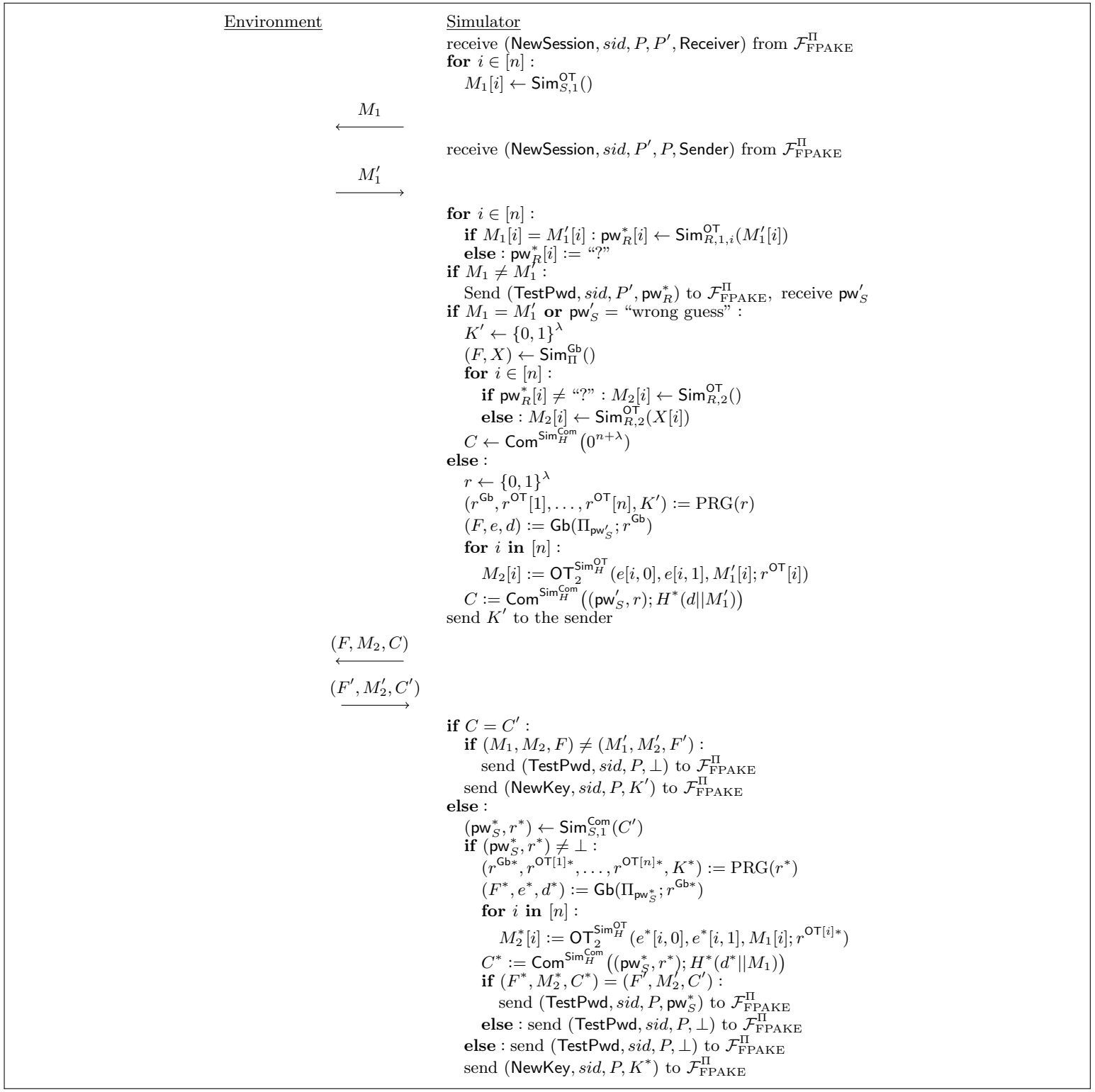

Figure 7: Simulator. The simulator uses $\mathsf{Sim}_H^{\mathsf{OT}}$ and $\mathsf{Sim}_H^{\mathsf{Com}}$ to respond to OT and commitment oracle queries, and honestly responds to H^* queries. If the simulator would send a pw_R^* containing any '?' as a password guess, it sends $\perp$ instead.

HYBRID$_2$(fig. 25): In HYBRID$_2$, we define a flag $\mathsf{correctguess}$ in step 2 as follows: if $M_1[i] = M_1'[i]$ for any $i \in [n]$ or $\Pi(\mathsf{pw}_R^*, \mathsf{pw}_S) = 0$ then $\mathsf{correctguess} = 0$, otherwise $\mathsf{correctguess} = 1$. (Intuitively, $\mathsf{correctguess}$ indicates whether the adversary modifies all of the OT messages $M_1[i]$, and these messages represent a correct password guess pw_R^*.) Then in step 3, if $\mathsf{correctguess} = 0$, $M_1' = M_1$, $C' = C$, and $d' = d$, then (pw_S^*, r^*) is set to be (pw_S, r) rather than $\mathsf{Decom}^{H_{\mathsf{Com}}}\big(C'; H^*(d'||M_1)\big)$. Under these conditions, correctness of the commitment scheme implies $\mathsf{Decom}^{H_{\mathsf{Com}}}\big(C'; H^*(d'||M_1)\big) = \mathsf{Decom}^{H_{\mathsf{Com}}}\big(C; H^*(d||M_1)\big) = (\mathsf{pw}_S, r)$, so HYBRID$_1$ and HYBRID$_2$ are identical.

HYBRID$_3$(fig. 26): In HYBRID$_3$, we make the following changes:

1. A dummy commitment $C \leftarrow \mathsf{Com}(0^{n+\lambda})$ is used in step 2 when $\mathsf{correctguess} = 0$ (as opposed to a real commitment in HYBRID$_2$).
2. As in the CIR proof, we add an abort condition: the hybrid aborts if $\mathsf{correctguess} = 0$ and the adversary queries $H^*(d||M_1')$. Call this event Query. We do not yet prove that $\Pr[\mathsf{Query}]$ is negligible, but will prove this in HYBRID$_9$ instead.
3. In step 3, if (pw_S^*, r^*) are set to be (pw_S, r) directly (i.e., $\mathsf{correctguess} = 0$ and $(M_1', C', d') = (M_1, C, d)$), we replace the consistency check $(F^*, M_2^*, C^*) = (F', M_2', C')$ with $(F^*, M_2^*) = (F', M_2')$ only.

If $(M_1', C', d') = (M_1, C, d)$, then C is not used in step 3, so using a dummy commitment is indistinguishable by commitment hiding (as long as the adversary does not query $H^*(d||M_1')$). Furthermore, in this case C^* is generated in exactly the same way as $C' = C$, so $C^* = C'$ and the consistency check for C^* is redundant. If $(M_1', C', d') \neq (M_1, C, d)$, HYBRID$_2$ and HYBRID$_3$ are indistinguishable for the following reasons:

- If $C' \neq C$, the dummy commitment does not matter because the adversary changed the commitment, so the hybrids are indistinguishable by commitment hiding
- If $C' = C$, then $(M_1', d') \neq (M_1, d)$ and thus $H(d'||M_1)$ is not the correct randomness for C, so the decommitment for $C' = C$ will fail.

HYBRID$_4$(fig. 27): In HYBRID$_4$, when $\mathsf{correctguess} = 0$ and $(C', M_1', d') = (C, M_1, d)$, the hybrid sends output to the receiver immediately and then stops rather then setting (pw_S^*, r^*) to be (pw_S, r) and then running the rest of the hybrid. Specifically, it sends K' (as generated in step 2) if $(M_2, F) = (M_2', F')$, and sends a random key otherwise. We also remove the exception to the consistency check — HYBRID$_4$ does not check that $(\mathsf{correctguess}, C', M_1', d') \neq (0, C, M_1, d)$ before sending $\perp$ to the receiver if $C' \neq C^*$. We argue that HYBRID$_3$ sends K' to the receiver if $(\mathsf{correctguess}, C', M_1', d', M_2', F') = (0, C, M_1, d, M_2, F)$, and sends a random key to the receiver if $(\mathsf{correctguess}, C', M_1', d') = (0, C, M_1, d)$ and $(M_2, F) \neq (M_2', F')$.

First, assume that $(\mathsf{correctguess}, C', M_1', d', M_2', F') = (0, C, M_1, d, M_2, F)$. In HYBRID$_3$, $(\mathsf{pw}_S^*, r^*) = (\mathsf{pw}_S, r)$, so F^* and d^* (which are outputs of $\mathsf{Gb}(\Pi_{\mathsf{pw}_S^*}; r^{\mathsf{Gb}*})$) are garbled identically to F and d (which are outputs of $\mathsf{Gb}(\Pi_{\mathsf{pw}_S}; r^{\mathsf{Gb}})$) and M_2^* (which is the output of $\mathsf{OT}_2^{\mathsf{Sim}_H^{\mathsf{OT}}}(e^*[i,0], e^*[i,1], M_1[i]; r^{\mathsf{OT}[i]*})$) is generated identically to M_2 (which is the output of $\mathsf{OT}_2^{\mathsf{Sim}_H^{\mathsf{OT}}}(e[i,0], e[i,1], M_1'[i]; r^{\mathsf{OT}}[i])$), so $(F', d', M_2') = (F^*, d^*, M_2^*)$. This is sufficient to show that the consistency checks pass in HYBRID$_3$ (because the $C' = C^*$ condition has an exception that is true in this case). So, HYBRID$_3$ sends K^* to the receiver, and $K^* = K'$.

Next, assume that $(\mathsf{correctguess}, C', M_1', d') = (0, C, M_1, d)$ and $(M_2', F') \neq (M_2, F)$. $(\mathsf{pw}_S^*, r^*) = (\mathsf{pw}_S, r)$ in this case, so $F = F^*$ and $M_2 = M_2^*$ for the reasons stated above. So, $(M_2', F') \neq (M_2, F)$ implies $(M_2', F') \neq (M_2^*, F^*)$. So, in this case, the consistency checks will fail and HYBRID$_3$ will send a random key to the receiver.

Finally, because HYBRID$_4$ stops before running the consistency checks when $(\mathsf{correctguess}, C', M_1', d') = (0, C, M_1, d)$, the code containing the exception is not reached when the exception is true, so it can be removed.

HYBRID$_5$(fig. 28): In HYBRID$_5$, we send a random key to the receiver when $(\mathsf{correctguess}, C) = (0, C')$ if $d \neq d'$ or $M_1' \neq M_1$. In this case, HYBRID$_4$ proceeds with step 3 normally. However, since $C = C'$ and the

commitment randomness is wrong, the decommitment will fail, so HYBRID_4 will also send a random key to the receiver.

HYBRID_6(fig. 29): In HYBRID_6, we send K' to the receiver if $(\mathsf{correctguess}, C, M_1, M_2, F) = (0, C', M_1', M_2', F')$ and $\Pi(\mathsf{pw}_R, \mathsf{pw}_S) = 1$. (In HYBRID_5, we sent K' to the receiver if $(\mathsf{correctguess}, C, M_1, M_2, F, d) = (0, C', M_1', M_2', F', d'))$. In other words, we replace the condition that $d = d'$ with a condition that $\Pi(\mathsf{pw}_R, \mathsf{pw}_S) = 1$. This check only occurs when $(M_1, M_2, F) = (M_1', M_2', F')$, so X is computed as the valid garbled encoding of pw_R earlier in step 3, and d' is the proper output of garbling Π_{pw_S}. So, we can apply perfect correctness to say that $d = d'$ if and only if $\Pi(\mathsf{pw}_R, \mathsf{pw}_S) = 1$.

HYBRID_7(fig. 30): In HYBRID_7, when $\mathsf{correctguess} = 0$, step 2 uses true randomness for K', Gb and OT (Rather than using the output of a PRG evaluated on r). Because r is not used for anything but a PRG seed (i.e., no longer in the commitment), the change is indistinguishable from the security of the PRG.

HYBRID_8(fig. 31): In HYBRID_8, when $\mathsf{correctguess} = 0$, step 2 runs $\mathsf{Sim}^{\mathsf{OT}}_{R,2}(e[i, \mathsf{pw}^*_R])$ to generate M_2 (rather than $\mathsf{OT}_2^{\mathsf{Sim}^{\mathsf{OT}}_H}(e[i,0], e[i,1], M_1'[i])$). This is indistinguishable by theorem 4.2.

HYBRID_9(fig. 32): In HYBRID_9, when $\mathsf{correctguess} = 0$, we generate an $X := \mathsf{En}(e, \mathsf{pw}^*_R)$ and use that directly as the input to $\mathsf{Sim}^{\mathsf{OT}}_{R,2}$. This is equivalent to the behavior of the previous hybrid. In addition, we remove the abort condition about the adversary querying the random oracle at $d \| M_1'$. When $\mathsf{correctguess} = 0$, at least one of the following is true:

- $\Pi(\mathsf{pw}^*_R, \mathsf{pw}_S) = 0$. In this case, the adversary sees F and the OT encoding of X, so garbling authenticity gives that it is hard to guess d.
- $M_1[i] = M_1'[i]$ for some $i \in [n]$. In this case, the adversary sees F and a partial OT encoding of X, so theorem 4.4 gives that it is hard to guess d.

Either way, $\Pr[\mathsf{Query}]$ in HYBRID_9 is negligible. (Note that d is not used anywhere in step 3). As the previous hybrids have been indistinguishable, the difference in $\Pr[\mathsf{Query}]$ was negligible between each hybrid, so $\Pr[\mathsf{Query}]$ in HYBRID_3 was negligible; so, HYBRID_2 and HYBRID_3 are indistinguishable.

HYBRID_{10}(fig. 33): In HYBRID_{10}, when $\mathsf{correctguess} = 0$, we generate (F, X) using $\mathsf{Sim}^{\mathsf{Gb}}_{\Pi}()$ (rather than generating it with $\mathsf{Gb}(\Pi_{\mathsf{pw}_S})$ and En). $\mathsf{correctguess} = 0$ implies that one of the following is true:

- $M_1[i] = M_1'[i]$ for some $i \in [n]$: In this case, this change is indistinguishable by obliviousness under partial inputs. (When $M_1[i] = M_1'[i]$, $\mathsf{pw}^*_R =$ '?', so M_2 is simulated without knowledge of $X[i]$; so this $X[i]$ is not in the adversary's view.)
- $\Pi(\mathsf{pw}^*_R, \mathsf{pw}_S) = 0$ and $M_1[i] \neq M_1'[i]$ for all $i \in [n]$: In this case, this change is indistinguishable by regular obliviousness.

HYBRID_{11}(fig. 34): In HYBRID_{11}, when $C = C'$ and $\mathsf{correctguess} = 1$, we send a random key to the receiver in step 3. $\mathsf{correctguess} = 1$ implies $M_1 \neq M_1'$, so in this case the randomness used to try to decommit C' will be wrong; Since $C' = C$, this implies the decommitment will fail, and a random key will be sent to the receiver (in HYBRID_{10}).

HYBRID_{12}(fig. 35): In HYBRID_{12}, when $C \neq C'$, if $\mathsf{Decom}^{\mathsf{Sim}^{\mathsf{Com}}_H}\left(C'; H^*(d' || M_1)\right) \neq \perp$, we use $\mathsf{Sim}^{\mathsf{Com}}_{S,1}(C')$ to extract the values (pw^*_S, r^*). Otherwise (if $\mathsf{Decom}^{\mathsf{Sim}^{\mathsf{Com}}_H}\left(C'; H^*(d' || M_1)\right) = \perp$) we set $(\mathsf{pw}^*_S, r^*) := \perp$. (In HYBRID_{11}, (pw^*_S, r^*) was the output of the decommitment). We also replace HCOM with $\mathsf{Sim}^{\mathsf{Com}}_H$. This change is indistinguishable by the UC-security of the commitment scheme against a corrupt simulator.

HYBRID$_{13}$(fig. 36): In HYBRID$_{13}$, when $C \neq C'$, we always use $\mathsf{Sim}^{\mathsf{Com}}_{S,1}(C')$ to extract the values (pw^*_S, r^*) (Rather than only doing so when $\mathsf{Decom}^{\mathsf{Sim}^{\mathsf{Com}}_H}\left(C'; H^*(d'||M_1)\right) \neq \perp$, and setting $(\mathsf{pw}^*_S, r^*) := \perp$ otherwise). We also only run the consistency checks if $\mathsf{Decom}^{\mathsf{Sim}^{\mathsf{Com}}_H}\left(C'; H^*(d'||M_1)\right) \neq \perp$ (in addition to requiring that $(\mathsf{pw}^*_S, r^*) \neq \perp$). This hybrid is logically equivalent to the previous.

HYBRID$_{14}$(fig. 37): In HYBRID$_{14}$, We replace the condition that $\mathsf{Decom}^{\mathsf{Sim}^{\mathsf{Com}}_H}\left(C'; H^*(d'||M_1)\right) \neq \perp$ required to run the consistency checks (i.e. the condition added in the previous hybrid) with the condition $\Pi(\mathsf{pw}^*_S, \mathsf{pw}_R) = 1$.Consider the following conditions (in both hybrids):

- $(F', M'_2, C', d') \neq (F^*, M^*_2, C^*, d^*)$ or $(\mathsf{pw}^*_S, r^*) = \perp$: regardless of what else happens, both hybrids will send a random key to the receiver.
- $(F', M'_2, C', d') = (F^*, M^*_2, C^*, d^*)$, $(\mathsf{pw}^*_S, r^*) \neq \perp$, and $\Pi(\mathsf{pw}^*_S, \mathsf{pw}_R) = 1$: In this case, C' is a commitment to (pw^*_S, r^*) generated with randomness $H^*(d^*||M_1) = H^*(d'||M_1)$. So, correctness of the commitment scheme implies $\mathsf{Decom}^{\mathsf{Sim}^{\mathsf{Com}}_H}\left(C'; H^*(d'||M_1)\right) \neq \perp$. So both hybrids will behave the same.
- $(F', M'_2, C', d') = (F^*, M^*_2, C^*, d^*)$, $(\mathsf{pw}^*_S, r^*) \neq \perp$, and $\Pi(\mathsf{pw}^*_S, \mathsf{pw}_R) = 0$: This case happens with negligible probability. F' is an honest garbling of $\Pi_{\mathsf{pw}^*_S}$ where d^* is the output label encoding true, so garbling authenticity gives that $\mathsf{Ev}(X^*, F') \neq d^*$ for the garbled encoding X^* of pw_R (i.e. since $\Pi(\mathsf{pw}^*_S, \mathsf{pw}_R) = 0$). So, $(F', M'_2, C') = (F^*, M^*_2, C^*)$, $(\mathsf{pw}^*_S, r^*) \neq \perp$, and $\Pi(\mathsf{pw}^*_S, \mathsf{pw}_R) = 0$ implies $d' \neq d^*$ with overwhelming probability.

HYBRID$_{15}$(fig. 38): In HYBRID$_{15}$, we remove the check that $d' = d^*$ before sending K^* to the receiver in step 3. As stated previously, $(F', M'_2, C') \neq (F^*, M^*_2, C^*)$, $(\mathsf{pw}^*_S, r^*) \neq \perp$, and $\Pi(\mathsf{pw}^*_S, \mathsf{pw}_R) = 0$ implies $d^* = d'$, so this check is redundant. Because d' is no longer used, the code generating it can be removed.

HYBRID$_{16}$(fig. 39): A condition in HYBRID$_{15}$ that has the form "$M_1[i] = M_1[i]$ for some i, or $\Pi(pw^*_R, pw_S) = 0$" is changed to "$M_1 = M'_1$ or the result of sending $(\mathsf{TestPwd}, sid, P', pw^*_R)$ to the functionality is *wrong guess*."

TestPwd replies with "wrong guess" in HYBRID$_{16}$ under the following circumstances:

- $\perp$ is sent as the password guess, which occurs if there are any '?' in pw^*_R, which occurs if $M_1[i] = M'_1[i]$ for some $i \in [n]$.
- $\perp$ is not sent as the password guess, but $\Pi(\mathsf{pw}^*_R, \mathsf{pw}_S) = 0$.

Thus, the conditions are logically equivalent in the two hybrids.

HYBRID$_{17}$(fig. 40): In HYBRID$_{17}$, when $C = C'$, we no longer send K to the receiver directly. Instead, when $C = C'$, we send $(\mathsf{TestPwd}, sid, P, \perp)$ to $\mathcal{F}^{\Pi}_{\text{FPAKE}}$ if $(M_1, M_2, F) \neq (M'_1, M'_2, F')$. We then send $(\mathsf{NewKey}, sid, P, K')$ to $\mathcal{F}^{\Pi}_{\text{FPAKE}}$ (regardless of whether or not $(M_1, M_2, F) \neq (M'_1, M'_2, F')$). Sending $(\mathsf{TestPwd}, sid, P, \perp)$ first ensures that NewKey always sends a random key to the receiver when $(M_1, M_2, F) \neq (M'_1, M'_2, F')$. If this TestPwd message is not sent, NewKey sends K' to the receiver if and only if $\Pi(\mathsf{pw}_R, \mathsf{pw}_S) = 1$ (and sends a random key otherwise). So, the conditions under which K' or a random key are sent to the reciever in HYBRID$_{16}$ and HYBRID$_{17}$ are the same.

HYBRID$_{18}$(fig. 41): In HYBRID$_{18}$, we make the following changes when $C \neq C'$:

- We no longer send a random key to the receiver if $\Pi(\mathsf{pw}^*_S, \mathsf{pw}_R) = 1$.
- If the commitment extraction produces $\perp$, we skip the consistency checks and send $(\mathsf{TestPwd}, sid, P, \perp)$ to $\mathcal{F}^{\Pi}_{\text{FPAKE}}$.
- At the end of the consistency checks, we send $(\mathsf{TestPwd}, sid, P, \mathsf{pw}^*_S)$ to $\mathcal{F}^{\Pi}_{\text{FPAKE}}$ if the checks pass and $(\mathsf{TestPwd}, sid, P, \perp)$ otherwise.

- Rather than sending a key to the receiver directly, we send $(\mathsf{NewKey}, sid, P, K^*)$ to $\mathcal{F}_{\text{FPAKE}}^{\Pi}$

If the consistency checks fail, or the commitment extraction produces $\perp$, a random key is always sent to the receiver in HYBRID_{17}; this is equivalent to guessing $\perp$ to $\mathsf{TestPwd}$ and then sending NewKey. The only remaining difference in behavior, then, is that HYBRID_{18} does not check that $\Pi(\mathsf{pw}_S^*, \mathsf{pw}_R) = 1$; However, this is checked implicitly in $\mathsf{TestPwd}$ and NewKey, so the behavior of the hybrids is the same.

Finally, HYBRID_{18} is identical to the simulation in the ideal world. □

References

[1] B. Barak, R. Canetti, Y. Lindell, R. Pass, and T. Rabin. Secure computation without authentication. In V. Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 361–377. Springer, Berlin, Heidelberg, Aug. 2005.

[2] D. Beaver. Correlated pseudorandomness and the complexity of private computations. In *28th ACM STOC*, pages 479–488. ACM Press, May 1996.

[3] M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In T. Yu, G. Danezis, and V. D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, Oct. 2012.

[4] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Computer Society Press, May 1992.

[5] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *ACM CCS 2017*, pages 1175–1191. ACM Press, Oct. / Nov. 2017.

[6] J. Bootle, S. H. Faller, J. Hesse, K. Hostáková, and J. Ottenhues. Generalized fuzzy password-authenticated key exchange from error correcting codes. In J. Guo and R. Steinfeld, editors, *ASIACRYPT 2023, Part VIII*, volume 14445 of *LNCS*, pages 110–142. Springer, Singapore, Dec. 2023.

[7] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

[8] R. Chatterjee, A. Athayle, D. Akhawe, A. Juels, and T. Ristenpart. pASSWORD tYPOS and how to correct them securely. In *2016 IEEE Symposium on Security and Privacy*, pages 799–818. IEEE Computer Society Press, May 2016.

[9] P.-A. Dupont, J. Hesse, D. Pointcheval, L. Reyzin, and S. Yakoubov. Fuzzy password-authenticated key exchange. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 393–424. Springer, Cham, Apr. / May 2018.

[10] A. Erwig, J. Hesse, M. Orlt, and S. Riahi. Fuzzy asymmetric password-authenticated key exchange. In S. Moriai and H. Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 761–784. Springer, Cham, Dec. 2020.

[11] T. K. Frederiksen, J. B. Nielsen, and C. Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 191–219. Springer, Berlin, Heidelberg, Apr. 2015.

[12] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In M. J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554. Springer, Berlin, Heidelberg, Aug. 1999.

[13] S. Garg, M. Mahmoody, D. Masny, and I. Meckler. On the round complexity of OT extension. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 545–574. Springer, Cham, Aug. 2018.

[14] Y. Huang, J. Katz, and D. Evans. Quid-Pro-Quo-tocols: Strengthening semi-honest protocols with dual execution. In *2012 IEEE Symposium on Security and Privacy*, pages 272–284. IEEE Computer Society Press, May 2012.

[15] Y. Huang, J. Katz, and D. Evans. Efficient secure two-party computation using symmetric cut-and-choose. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 18–35. Springer, Berlin, Heidelberg, Aug. 2013.

[16] M. Jawurek, F. Kerschbaum, and C. Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 2013*, pages 955–966. ACM Press, Nov. 2013.

[17] J. Katz, S. Ranellucci, M. Rosulek, and X. Wang. Optimizing authenticated garbling for faster secure two-party computation. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 365–391. Springer, Cham, Aug. 2018.

[18] V. Kolesnikov, P. Mohassel, B. Riva, and M. Rosulek. Richer efficiency/security trade-offs in 2PC. In Y. Dodis and J. B. Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 229–259. Springer, Berlin, Heidelberg, Mar. 2015.

[19] Y. Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 1–17. Springer, Berlin, Heidelberg, Aug. 2013.

[20] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In M. Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 52–78. Springer, Berlin, Heidelberg, May 2007.

[21] Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *Journal of Cryptology*, 25(4):680–722, Oct. 2012.

[22] D. Masny and P. Rindal. Endemic oblivious transfer. In L. Cavallaro, J. Kinder, X. Wang, and J. Katz, editors, *ACM CCS 2019*, pages 309–326. ACM Press, Nov. 2019.

[23] I. McQuoid, M. Rosulek, and L. Roy. Minimal symmetric PAKE and 1-out-of-N OT from programmable-once public functions. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *ACM CCS 2020*, pages 425–442. ACM Press, Nov. 2020.

[24] I. McQuoid, M. Rosulek, and J. Xu. How to tolerate typos in strong asymmetric PAKE. In *CRYPTO 2025, Part III*, LNCS, pages 610–641. Springer, Cham, Aug. 2025.

[25] P. Mohassel and M. Franklin. Efficiency tradeoffs for malicious two-party computation. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 458–473. Springer, Berlin, Heidelberg, Apr. 2006.

[26] P. Mohassel and B. Riva. Garbled circuits checking garbled circuits: More efficient and secure two-party computation. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 36–53. Springer, Berlin, Heidelberg, Aug. 2013.

[27] C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Berlin, Heidelberg, Aug. 2008.

[28] T. Pongmorrakot and R. Chatterjee. tPAKE: typo-tolerant password-authenticated key exchange. In L. Batina, S. Picek, and M. Mondal, editors, *Security, Privacy, and Applied Cryptography Engineering - 10th International Conference, SPACE 2020, Kolkata, India, December 17-21, 2020, Proceedings*, volume 12586 of *Lecture Notes in Computer Science*, pages 3–24. Springer, 2020.

[29] M. Rosulek and L. Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In T. Malkin and C. Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 94–124, Virtual Event, Aug. 2021. Springer, Cham.

[30] a. shelat and C.-H. Shen. Fast two-party secure computation with minimal assumptions. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 2013*, pages 523–534. ACM Press, Nov. 2013.

[31] X. Wang, S. Ranellucci, and J. Katz. Authenticated garbling and efficient maliciously secure two-party computation. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *ACM CCS 2017*, pages 21–37. ACM Press, Oct. / Nov. 2017.

[32] A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, Oct. 1986.

[33] S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Berlin, Heidelberg, Apr. 2015.

A Proofs from Section 4.2

Proof of Lemma 4.2. Consider an adversary $\mathcal{A}$ attempting to distinguish the two games shown in Lemma 4.2. Let $\mathcal{B}$ be a reduction algorithm that does the following: It acts as a minimal wrapper around $\mathcal{A}$, relaying all inputs/outputs between $\mathcal{A}$ and $\mathcal{B}$'s game and $\mathcal{B}$'s oracle — with one exception: When $\mathcal{B}$ receives M_2 from its game, it ignores this value and instead computes $M_2^* \leftarrow \mathsf{OT}_2^{H_{\mathsf{OT}}}(v_0, v_1, M_1')$ and passes M_2^* to $\mathcal{A}$ instead of M_2.

Now imagine $\mathcal{B}$ playing the game of distinguishing the games in Definition 4.1. The result will be the following two games, which must be indistinguishable by Definition 4.1.

G_1:

$b \leftarrow \mathcal{A}^H()$
$M_1 \leftarrow \mathsf{OT}_1^{H_{\mathsf{OT}}}(b)$
$(v_0, v_1, M_1') \leftarrow \mathcal{A}^H(M_1)$
$M_2 \leftarrow \mathsf{OT}_2^{H_{\mathsf{OT}}}(v_0, v_1, M_1')$
$\boxed{M_2^* \leftarrow \mathsf{OT}_2^{H_{\mathsf{OT}}}(v_0, v_1, M_1')}$
$M_2' \leftarrow \mathcal{A}^{H_{\mathsf{OT}}}(\boxed{M_2^*})$
$v \leftarrow \mathsf{OT}_3^{H_{\mathsf{OT}}}(\sigma, M_2')$
output $\mathcal{A}^{H_{\mathsf{OT}}}(v)$

$\approx$

G_2:

$b \leftarrow \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}()$
$M_1 \leftarrow \mathsf{Sim}_{S,1}^{\mathsf{OT}}()$
$(v_0, v_1, M_1') \leftarrow \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(M_1)$
if $M_1 = M_1'$:
  $M_2 \leftarrow \mathsf{Sim}_{R,2}^{\mathsf{OT}}()$
else:
  $b' \leftarrow \mathsf{Sim}_{R,1}^{\mathsf{OT}}(M_1')$
  $M_2 \leftarrow \mathsf{Sim}_{R,2}^{\mathsf{OT}}(v_{b'})$
$\boxed{M_2^* \leftarrow \mathsf{OT}_2^{\mathsf{Sim}_H^{\mathsf{OT}}}(v_0, v_1, M_1')}$
$M_2' \leftarrow \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(\boxed{M_2^*})$
if $(M_1, M_2) = (M_1', M_2')$:
  output $\mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(v_b)$
else:
  $(v_0', v_1') \leftarrow \mathsf{Sim}_{S,2}^{\mathsf{OT}}(M_2')$
  output $\mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(v_b')$

The boxed parts are the changes introduced by the $\mathcal{B}$ wrapper.

Note: G_1 is identically distributed to the "real" game of Definition 4.1, against adversary $\mathcal{A}$. In other words, the choice of M_2^* vs M_2 is immaterial, since they are distributed identically. Hence, G_1 is indistinguishable from the "ideal" game of Definition 4.1, against adversary $\mathcal{A}$. This is also the "right" game in Lemma 4.2.

Now, focusing on G_2 above: It differs from the "left" game in Lemma 4.2 only in one respect. G_2 has an if statement for condition $(M_1, M_2) = (M_1', M_2')$, where M_2 is a value never given to the adversary. In Lemma 4.2, the analogous condition uses the M_2 value actually given to the adversary. It suffices to show that G_2 is indistinguishable from one whose if-condition is $(M_1, \boxed{M_2^*}) = (M_1', M_2')$, because this would be identical behavior to the game in Lemma 4.2 after renaming variables.

Suppose the condition $(M_1, M_2^*) = (M_1', M_2')$ is true in G_2 above. Then the game will enter the else-branch. Since we know that M_2^* was generated as an honest OT message on inputs v_0, v_1, it is with overwhelming probability that the simulator $\mathsf{Sim}_{S,2}^{\mathsf{OT}}$ will extract $(v_0', v_1') = (v_0, v_1)$. (If this were not true, then even a semi-honest adversary would break the simulation). Then the game will give $v_b' = v_b$ to the adversary in the last line. This is exactly the same behavior of the if-branch! Thus, G_2 is indistinguishable from the "left" game of Lemma 4.2 and the lemma is proved. □

The proof of Lemma 4.3 is similar:

Proof of Lemma 4.3. Consider an adversary $\mathcal{A}$ attempting to distinguish the two games shown in Lemma 4.3. Let $\mathcal{B}$ be a reduction algorithm that does the following: It acts as a minimal wrapper around $\mathcal{A}$, relaying all inputs/outputs between $\mathcal{A}$ and $\mathcal{B}$'s game and $\mathcal{B}$'s oracle — with one exception: After $\mathcal{A}$ receives M_1, it outputs (v_0, v_1, M_1', r). $\mathcal{B}$ stores r and forwards only (v_0, v_1, M_1') to its external game. After receiving M_2 from its external game, it throws this value away and instead generates $M_2^* := \mathsf{OT}_2^{H_{\mathsf{OT}}}(v_0, v_1, M_1'; r)$ and gives M_2^* to $\mathcal{A}$ instead of M_2.

Now imagine $\mathcal{B}$ playing the game of distinguishing the games in Definition 4.1. The result will be the following two games, which must be indistinguishable by Definition 4.1.

G_1:

$b \leftarrow \mathcal{A}^H()$
$M_1 \leftarrow \mathsf{OT}_1^{H_{\mathsf{OT}}}(b)$
$(v_0, v_1, M_1', r) \leftarrow \mathcal{A}^H(M_1)$
$M_2 \leftarrow \mathsf{OT}_2^{H_{\mathsf{OT}}}(v_0, v_1, M_1')$
$\boxed{M_2^* := \mathsf{OT}_2^{H_{\mathsf{OT}}}(v_0, v_1, M_1'; r)}$
$M_2' \leftarrow \mathcal{A}^{H_{\mathsf{OT}}}(\boxed{M_2^*})$
$v \leftarrow \mathsf{OT}_3^{H_{\mathsf{OT}}}(\sigma, M_2')$
output $\mathcal{A}^{H_{\mathsf{OT}}}(v)$

$\approx$

G_2:

$b \leftarrow \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}()$
$M_1 \leftarrow \mathsf{Sim}_{S,1}^{\mathsf{OT}}()$
$(v_0, v_1, M_1', r) \leftarrow \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(M_1)$
if $M_1 = M_1'$:
  $M_2 \leftarrow \mathsf{Sim}_{R,2}^{\mathsf{OT}}()$
else:
  $b' \leftarrow \mathsf{Sim}_{R,1}^{\mathsf{OT}}(M_1')$
  $M_2 \leftarrow \mathsf{Sim}_{R,2}^{\mathsf{OT}}(v_{b'})$
$\boxed{M_2^* := \mathsf{OT}_2^{\mathsf{Sim}_H^{\mathsf{OT}}}(v_0, v_1, M_1'; r)}$
$M_2' \leftarrow \mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(\boxed{M_2^*})$
if $(M_1, M_2) = (M_1', M_2')$:
  output $\mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(v_b)$
else:
  $(v_0', v_1') \leftarrow \mathsf{Sim}_{S,2}^{\mathsf{OT}}(M_2')$
  output $\mathcal{A}^{\mathsf{Sim}_H^{\mathsf{OT}}}(v_b')$

The boxed parts are the changes introduced by the $\mathcal{B}$ wrapper.

Note that G_1 is identically distributed to the "left" game in Lemma 4.3, after accounting for the fact that M_2 in G_1 is not used, and after renaming M_2^*.

Focusing on G_2, as in the previous proof, it differs only from the "right" game in Lemma 4.3 in its logic in the final if-statement. In G_2 we compare M_2' to the M_2 value which was never shown to the adversary. In the "right" game in Lemma 4.3, we compare M_2' to the M_2 value actually shown to the adversary.

The reasoning here is essentially the same as the previous proof. But here we use the fact that the OT protocol has perfect correctness. With perfect correctness, the simulator must extract v_0, v_1 from M_2^*, even though M_2^* was generated by running the protocol with adversarially provided randomness. □

B Non-Malleable OT Instantiation

B.1 Preliminaries

Here we briefly review the PVW [27] protocol for oblivious transfer. Their construction is based on a primitive called **dual-mode encryption.** We refer the reader to their paper for full details, and give only a high-level summary:

Overview. The system is governed by a common reference string (CRS) that can be in one of two modes: *messy mode* or *decryption mode*. CRSs are generated with an associated trapdoor, which serves different purposes in the two modes. Importantly, the two modes are indistinguishable.

With respect to a CRS, we have a relatively standard public-key encryption scheme. However, we have a non-standard notion of *encryption branches.* Each public key has 2 corresponding branches, and when sampling a public key it is possible to designate one of the branches as a *decryptable* branch. Ciphertexts are encrypted under a public key but also a choice of branch. A secret key can only decrypt ciphertexts addressed to its decryptable branch. But the public key leaks nothing about which branch is decryptable.

In *messy mode*, the non-decryptable branch is called a *messy* branch. Ciphertexts encrypted to the messy branch hide their plaintext, information theoretically, even to the secret key holder. In messy mode, the trapdoor of the CRS allows one to determine the decryptable branch of any public key, even adversarially generated ones.

In *decryption mode*, the trapdoor of the CRS allows one to generate a public key whose branches are *both* decryptable (along with secret keys for both branches).

Syntax. The syntax of a dual-mode encryption is summarized below:

- $\mathsf{SetupMessy}() \to (crs, \tau)$: generates a messy-mode CRS with associated trapdoor τ
- $\mathsf{SetupDec}() \to (crs, \tau)$: generates a decryption-mode CRS with associated trapdoor τ
- $\mathsf{KeyGen}(crs, b) \to (sk, pk)$: given a CRS and branch selection b, generates a keypair where sk can decrypt ciphertexts addressed to branch b.
- $\mathsf{Enc}(crs, pk, b, m) \to c$: encrypts plaintext m to branch b of public key pk.
- $\mathsf{Dec}(sk, b, c) \to m$: decrypts a ciphertext addressed to branch b.
- $\mathsf{FindMessy}(\tau, pk) \to b$: given the trapdoor for a messy-mode CRS and a public key pk, determines which branch of pk is messy.
- $\mathsf{TrapKeyGen}(\tau) \to (sk_0, sk_1, pk)$: given the trapdoor of a decryption-mode CRS, generates a public key along with secret keys that can decrypt both branches.

The protocol. The PVW protocol is described in Figure 8.

B.2 Preliminaries

Our construction relies on the following variant of the oracle-Diffie-Hellman assumption from [5]:

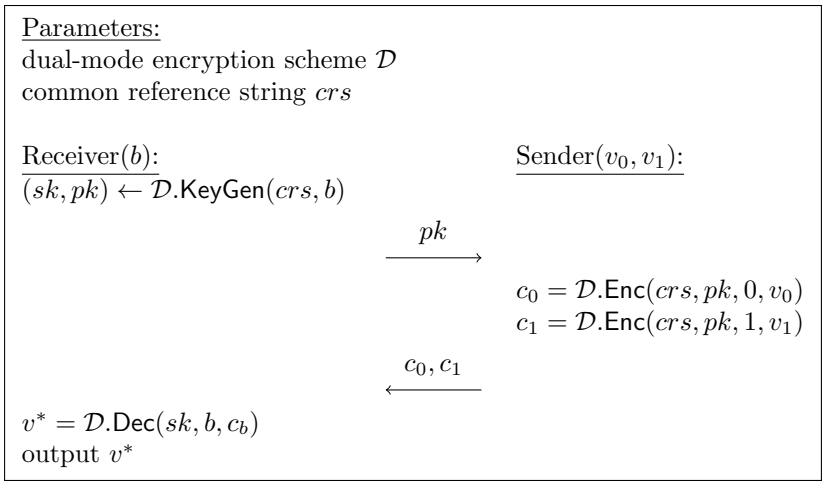

Figure 8: The PVW protocol.

Definition B.1. *Let $\mathbb{G}$ be a cyclic group with generator g and order n and let H be a hash function with range $\{0,1\}^\lambda$. Define the function $\mathcal{O}_s(X) = H(X^s)$.*

*The **2-oracle DH (2ODH)** assumption [5] holds with respect to $\mathbb{G}$ and H if for all efficient adversaries $\mathcal{A}$, the following two games are indistinguishable:*

$a, b \leftarrow \mathbb{Z}_n$ $K = H(g^{ab})$ *output* $\mathcal{A}^{\mathcal{O}_a, \mathcal{O}_b}(g^a, g^b, K)$	$a, b \leftarrow \mathbb{Z}_n$ $K \leftarrow \{0,1\}^\lambda$ *output* $\mathcal{A}^{\mathcal{O}_a, \mathcal{O}_b}(g^a, g^b, K)$

where the games additionally abort if $\mathcal{A}$ queries $\mathcal{O}_a$ on input g^b or queries $\mathcal{O}_b$ on input g^a.

Our security proof only needs a weaker version of 2ODH in which $\mathcal{A}$ makes *at most 1 query* total to its oracles. This limited variant of 2ODH holds assuming only the standard CDH assumption, if H is a random oracle, although the reduction is not tight.

B.3 Our non-malleable variant

The PVW protocol is not immediately secure in a man-in-the-middle setting. One particular challenge is that the CRS can only be in decryption mode or noisy mode, and each mode is only useful for corrupt sender or corrupt receiver, but not both. But a man-in-the-middle attacker is playing the role of corrupt receiver and corrupt sender, simultaneously, against different honest parties.

Our approach to achieve non-malleability is to add a "wrapper" around the PVW protocol. There are 2 main features to our wrapper:

- The parties will perform a (hashed) Diffie-Hellman key exchange in parallel to the PVW protocol. Thus, when the OT sender (who speaks 2nd) wants to send its OT message, it already knows the DH session key, and can use the session key to encrypt its PVW message. (Think of the second PVW message as being sent through a secure channel.) This prevents a man-in-the-middle from succesfully tampering with the second PVW message in certain ways.

- Our wrapper is in the random oracle model, and the parties will use $H(R)$ as the CRS for PVW, where R is the OT receiver's DH message. This modification is useful because the simulator can arrange for $H(R)$ to be a decryption-mode CRS, but if a man-in-the-middle adversary changes R in any way — say, by sending $R' \neq R$ — the simulator can arrange for $H(R')$ to be a messy-mode CRS.

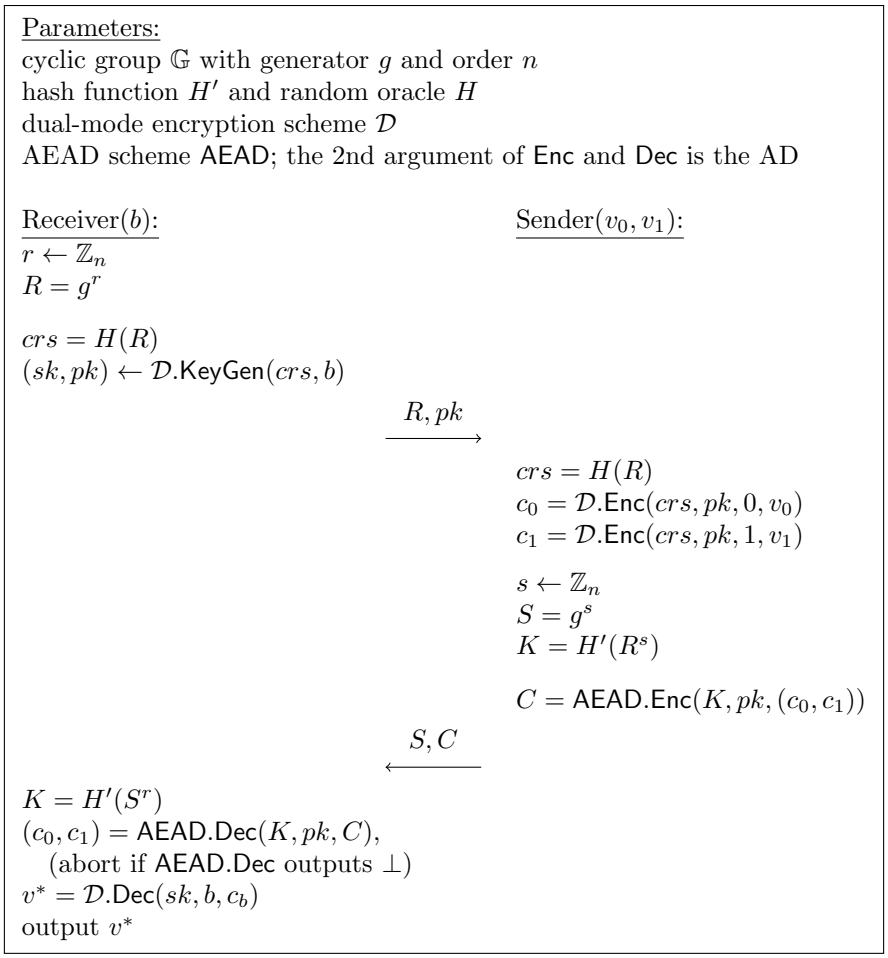

Figure 9: Our non-malleable variant of the PVW protocol.

The final point does not help if the man-in-the-middle preserves R but tampers with the receiver's PVW protocol message. We handle this case by making this PVW protocol message part of the authenticated data for encrypting the sender's PVW response (now encrypted under AEAD).

Theorem B.2. *The protocol in is a non-malleable OT protocol if (1) the 2ODH assumption holds with respect to $\mathbb{G}$ and H'. (2) $\mathcal{D}$ is a dual-mode encryption with pseudorandom CRS; (3) AEAD is a secure AEAD scheme; (4) H is a random oracle.*

Proof. Recall the non-malleability attack scenario from Definition 4.1. The adversary is acting as man-in-the-middle between two honest parties running the OT protocol — the receiver with input b and sender with input v_0, v_1. The situation should be indistinguishable from one involving a collection of simulators that do the following:

- The simulator simulates the receiver's first protocol message M_1 without knowledge of b.
- The adversary delivers some M_1' to the sender. If $M_1' \neq M_1$ then the simulator can extract b' and simulate the sender's message M_2 given just $v_{b'}$. Otherwise $M_1' = M_1$ and the simulator simulates M_2 given nothing.
- The adversary delivers some M_2' to the receiver. If $(M_1, M_2) = (M_1', M_2')$, the receiver's output is simulated as v_b. Otherwise, the simulator can extract v_0', v_1' and the receiver's output is v_b'.

We sketch the simulator behavior for our protocol. In Definition 4.1 the simulator is written as separate algorithms that all share common state. Below, for simplicity we write the simulator as a monolithic entity.

1. The simulator simulates H to output messy-mode CRSs. In other words, in response to every fresh random oracle query $H(x)$, the simulator runs $(c, \tau) \leftarrow \mathsf{SetupMessy}()$, and lets c be the output. It stores τ as $\tau(x)$ for use later.

2. To simulate the honest receiver's first message:

   - The simulator initially samples $R = g^r$, $S = g^s$ and programs the random oracle so that $H(R)$ is a decryption-mode CRS (unlike all other outputs of H) with trapdoor τ^*.
   - The simulator uses τ^* to run $\mathsf{TrapKeyGen}(H(R))$ and generate a trapdoored public key pk, with two corresponding decryption keys sk_0, sk_1..
   - (R, pk) is the simulated first OT message.

3. The adversary produces its own (R', pk') and if these are not identical to (R, pk), the simulator is tasked to extract a choice bit b'.

   - If $R = R'$ but $pk \neq pk'$, the simulator can determine b' arbitrarily. Looking ahead, the honest sender's response will be meaningless to the adversary (having been encrypted with g^{rs}), so the extraction is moot.
   - If $R \neq R'$ then $H(R')$ is a messy CRS, so the simulator can identify a messy branch of pk' using its CRS trapdoor $\tau(R')$. The simulator designates b' as the non-messy branch.

4. The simulator must simulate the honest sender's response, consisting of S and C. S is already chosen above (a random DH message). C is produced as follows:

   - If the adversary has not tampered with the first protocol message — i.e., if $(R, pk) = (R', pk')$ — then the response must be simulated with no information at all. This done with S sampled uniformly above, and by simulating C as a dummy AEAD ciphertext. This is a valid simulation because in this case the AEAD ciphertext would have been encrypted with $H'(g^{rs})$, which is pseudorandom to the adversary.
   - Otherwise, $(R, pk) \neq (R', pk')$, and the response must be simulated based only on $v_{b'}$, one of the honest sender's two inputs. The simulator generates ciphertext $c_{b'}$ honestly (an encryption of $v_{b'}$). But since pk' is messy at branch $1 - b'$, it can let $c_{1-b'}$ be a dummy ciphertext. The simulator then generates C by encrypting (c_0, c_1) under encryption key $H'((R')^s)$.

5. Now the adversary produces its own (S', C'), and if $(R, pk, S, C) \neq (R', pk', S', C')$ then the simulator is tasked with extracting OT payloads.

   - If $(R, S) = (R', S')$, the simulator can extract arbitrarily; the choice does not matter. To see why, note that in this case, the honest sender would previously generate C as an AEAD ciphertext with AD pk, under key $H'(g^{rs})$. This is the only ciphertext the adversary sees under this key. Later, the honest receiver would decrypt C' with AD pk' under the same key $H'(g^{rs})$. But in this case we must have $(C, pk) \neq (C', pk')$, so the honest receiver's decryption would fail by the guarantee of AEAD.
   - If $S = S'$ and $R \neq R'$, then again the simulator's extraction strategy is irrelevant. In this case the honest receiver would again decrypt C' with AD pk' under the same key $H'(g^{rs})$. But in this case, the adversary would not have seen any ciphertext encrypted under this key — instead, the honest sender would have encrypted its C under key $H'((R')^s)$. Again, by the AEAD guarantee, the honest receiver's decryption would have failed.

- The only other case is $S \neq S'$. In this case, the simulator decrypts C' under key $H'((S')^r)$, as the honest receiver would do, to obtain c'_0, c'_1. It can decrypt these with sk_0, sk_1, respectively (recall that the corresponding CRS is decrypt-mode and this pk was generated with the CRS trapdoor to have both branches decryptable); the results (v'_0, v'_1) are what the simulator extracts.

Note that the analysis requires $H'(g^{rs})$ to be pseudorandom, even while the simulator uses the values $H'((S')^r)$ or $H'((R')^s)$, for $S' \neq g^s$ or $R' \neq g^r$. This is exactly the guarantee given by the 2ODH assumption. □

C Hybrids from Security Proofs

C.1 CIR-2PC Corrupt Sender Hybrids

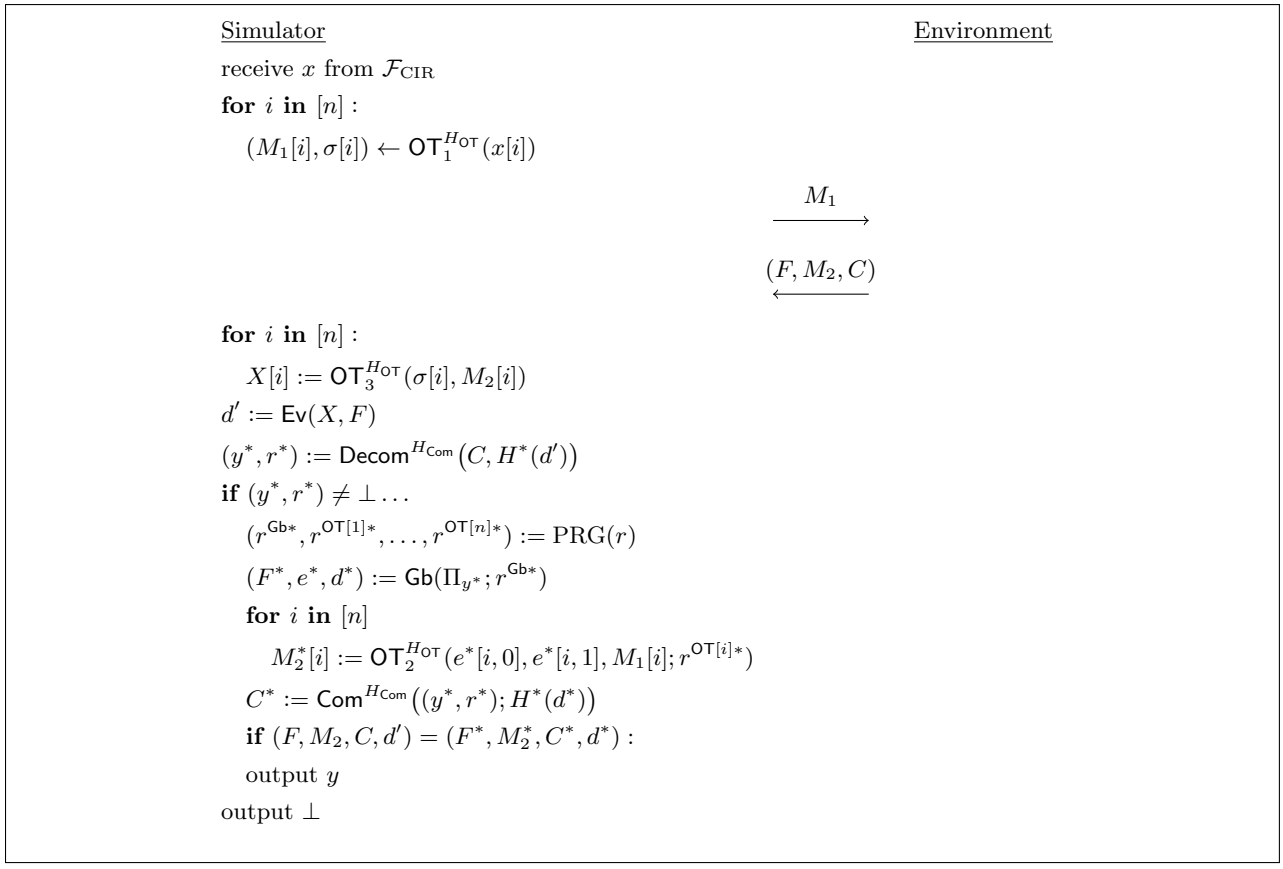

Figure 10: HYBRID_0 in the security proof of the CIR-2PC protocol, for a corrupt sender.

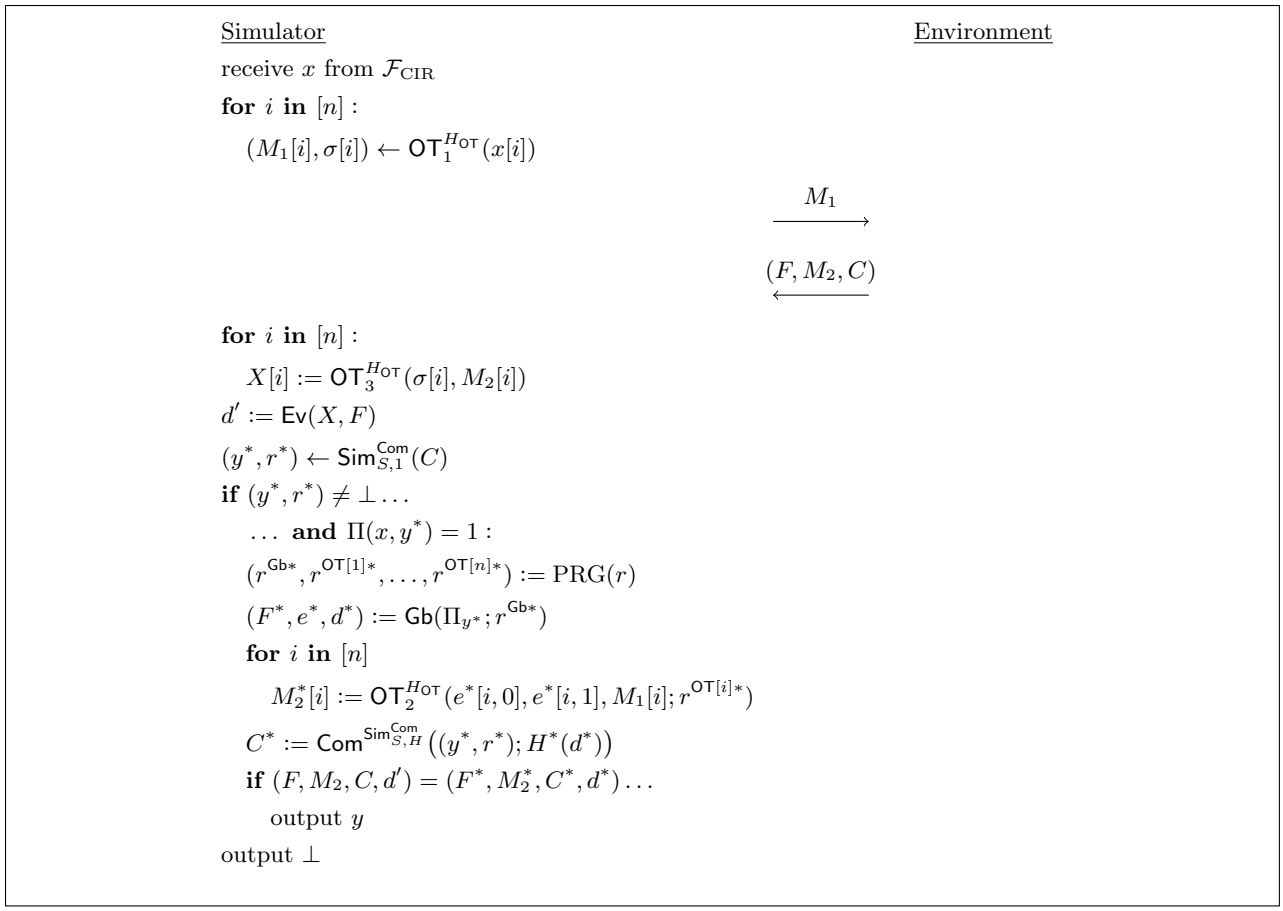

Figure 11: HYBRID$_1$ in the security proof of the CIR-2PC protocol, for a corrupt sender.

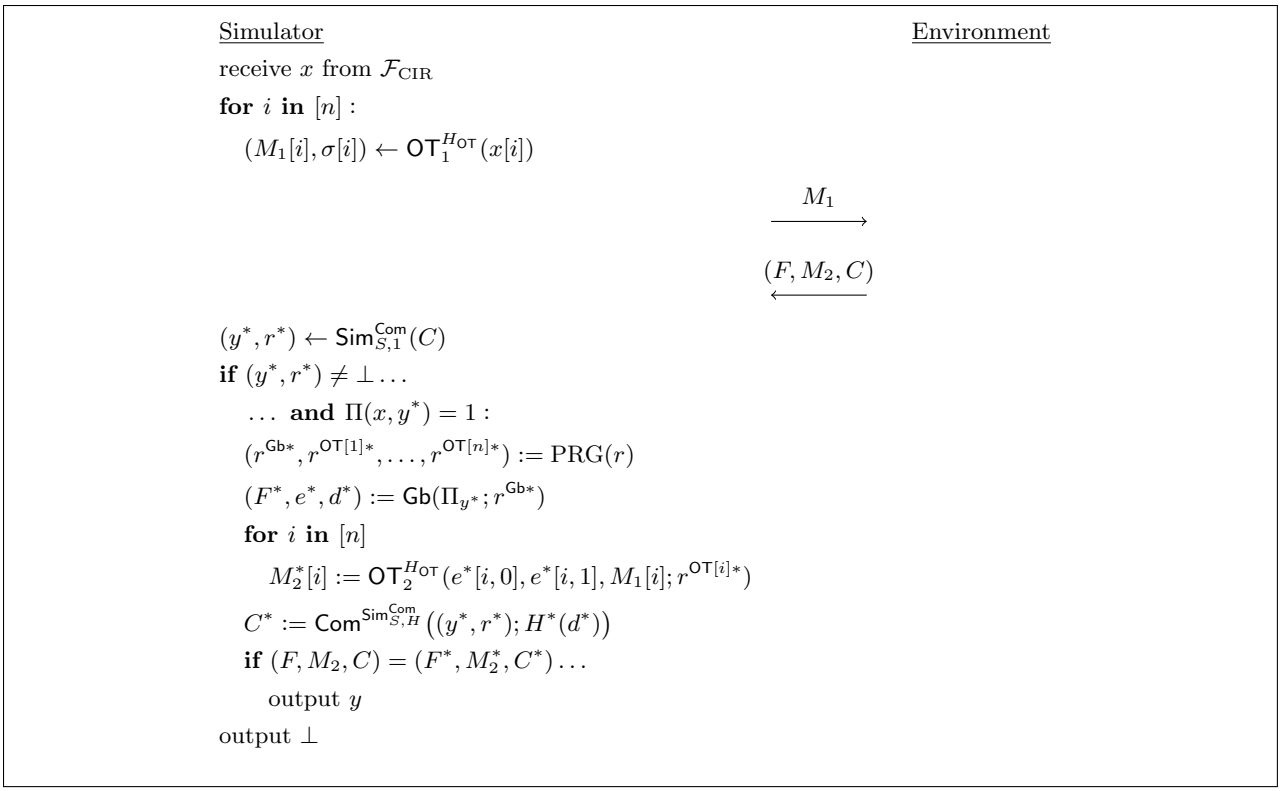

Figure 12: HYBRID$_2$ in the security proof of the CIR-2PC protocol, for a corrupt sender.

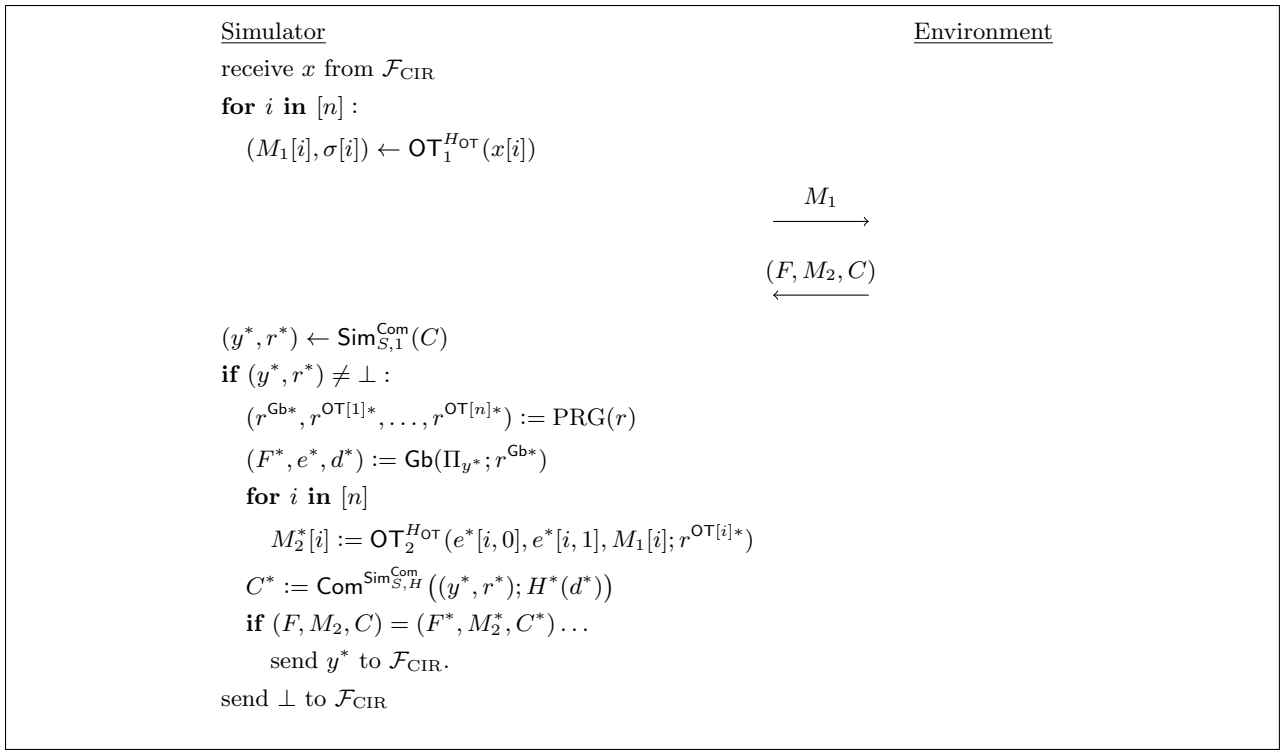

Figure 13: HYBRID$_3$ in the security proof of the CIR-2PC protocol, for a corrupt sender.

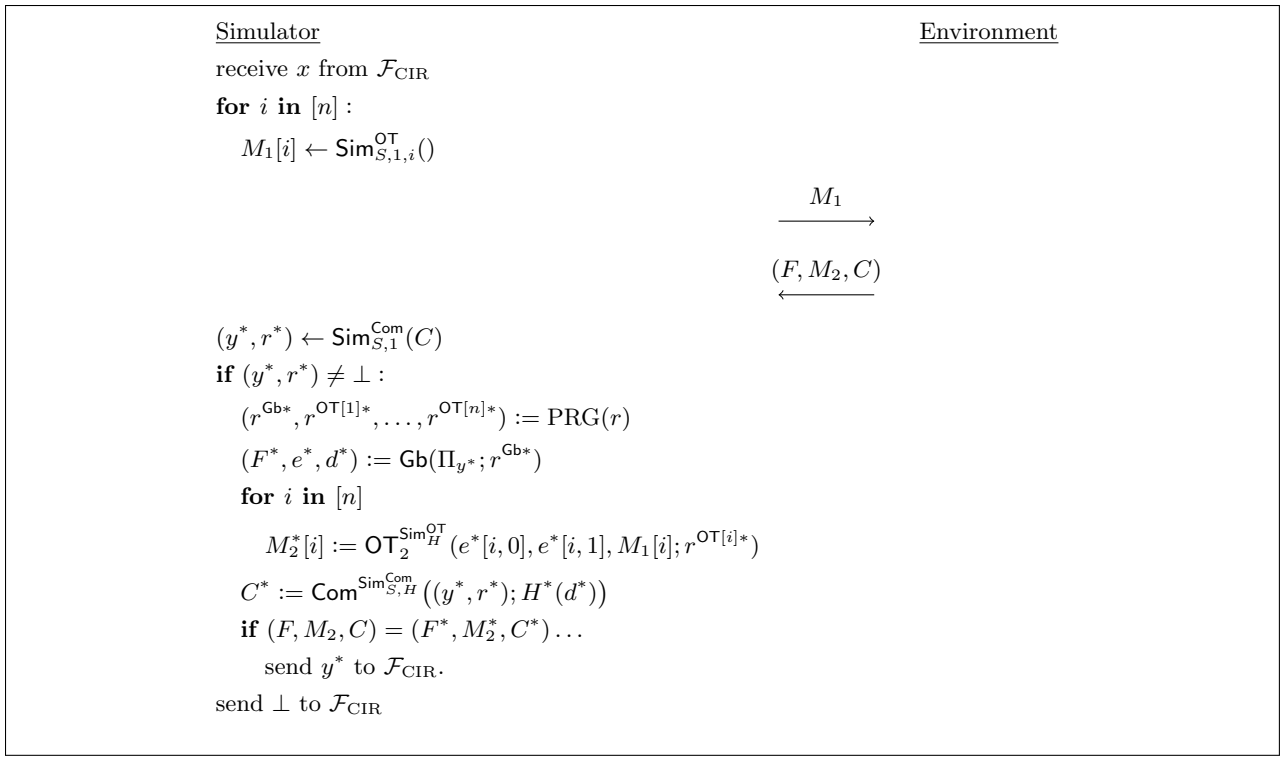

Figure 14: HYBRID$_4$ in the security proof of the CIR-2PC protocol, for a corrupt sender.

C.2 CIR-2PC Corrupt Receiver Hybrids

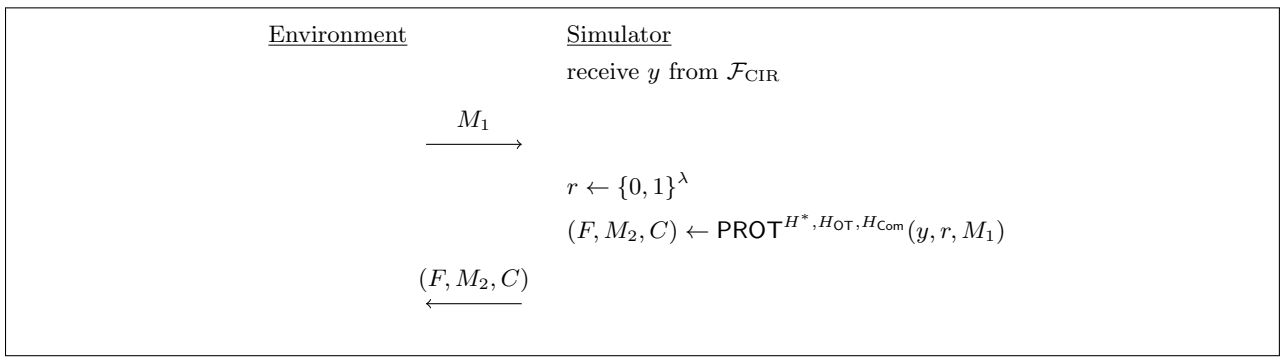

Figure 15: HYBRID$_0$ in the security proof of the CIR-2PC protocol, for a corrupt receiver.

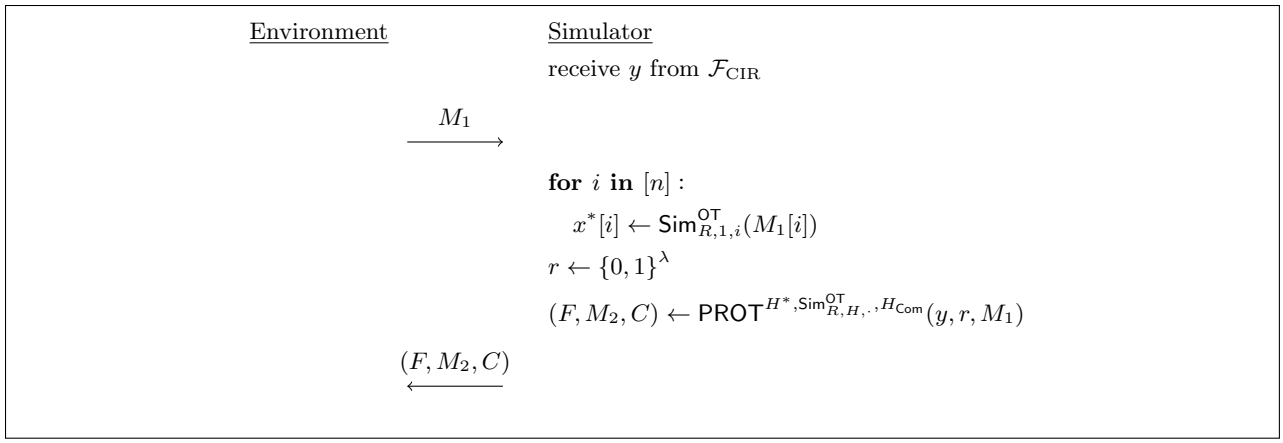

Figure 16: HYBRID$_1$ in the security proof of the CIR-2PC protocol, for a corrupt receiver.

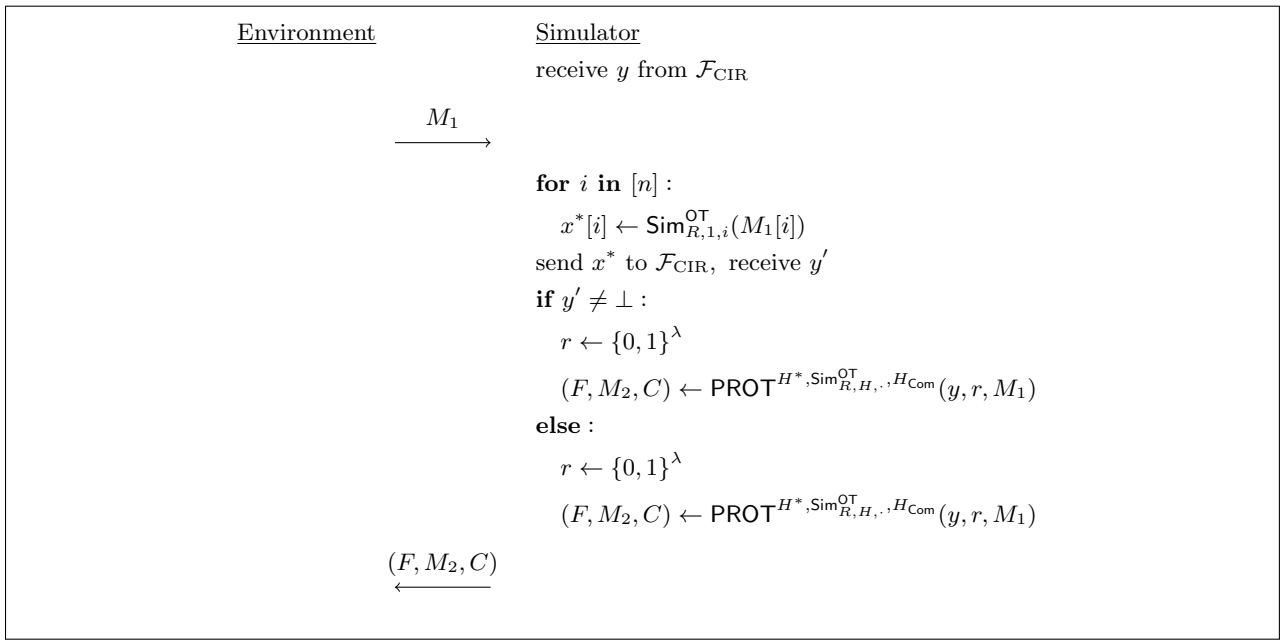

Figure 17: HYBRID$_2$ in the security proof of the CIR-2PC protocol, for a corrupt receiver.

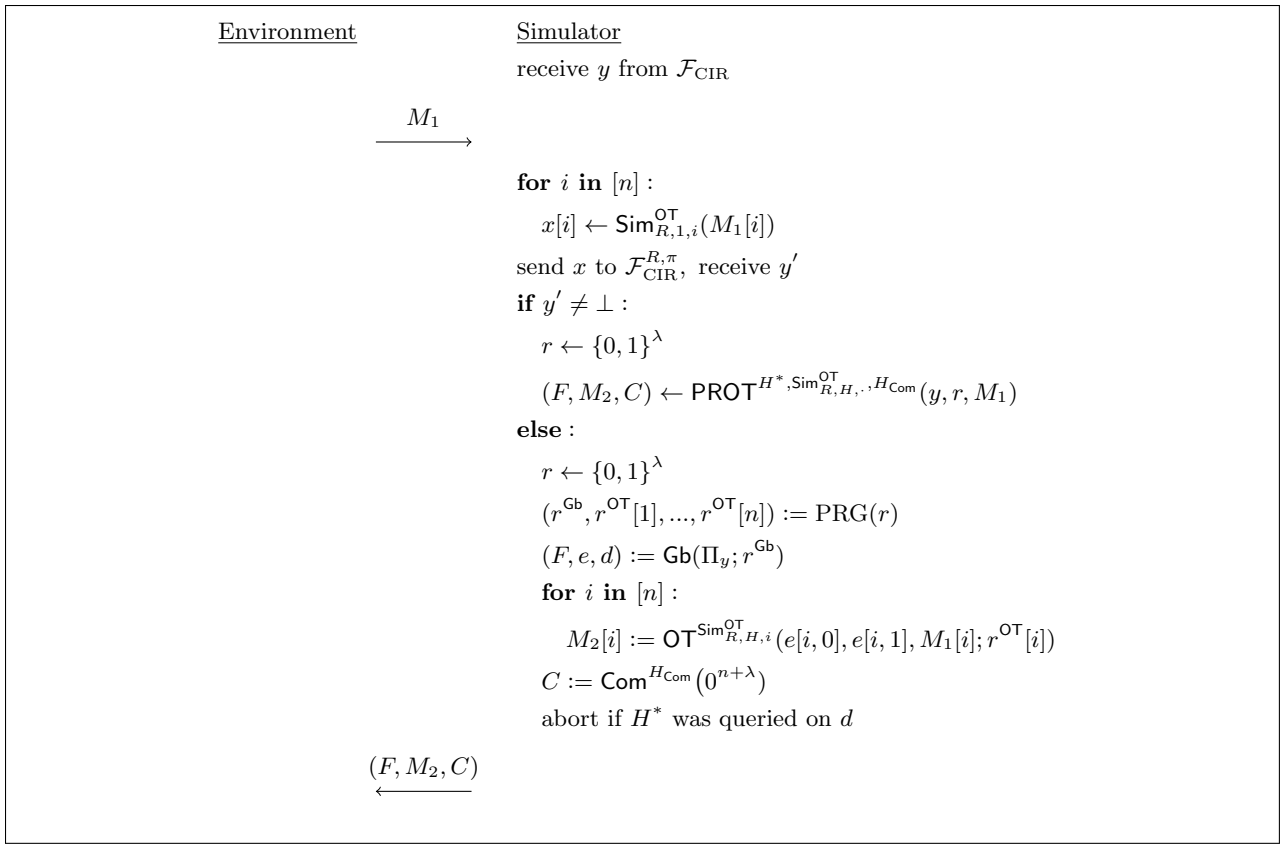

Figure 18: HYBRID$_3$ in the security proof of the CIR-2PC protocol, for a corrupt receiver.

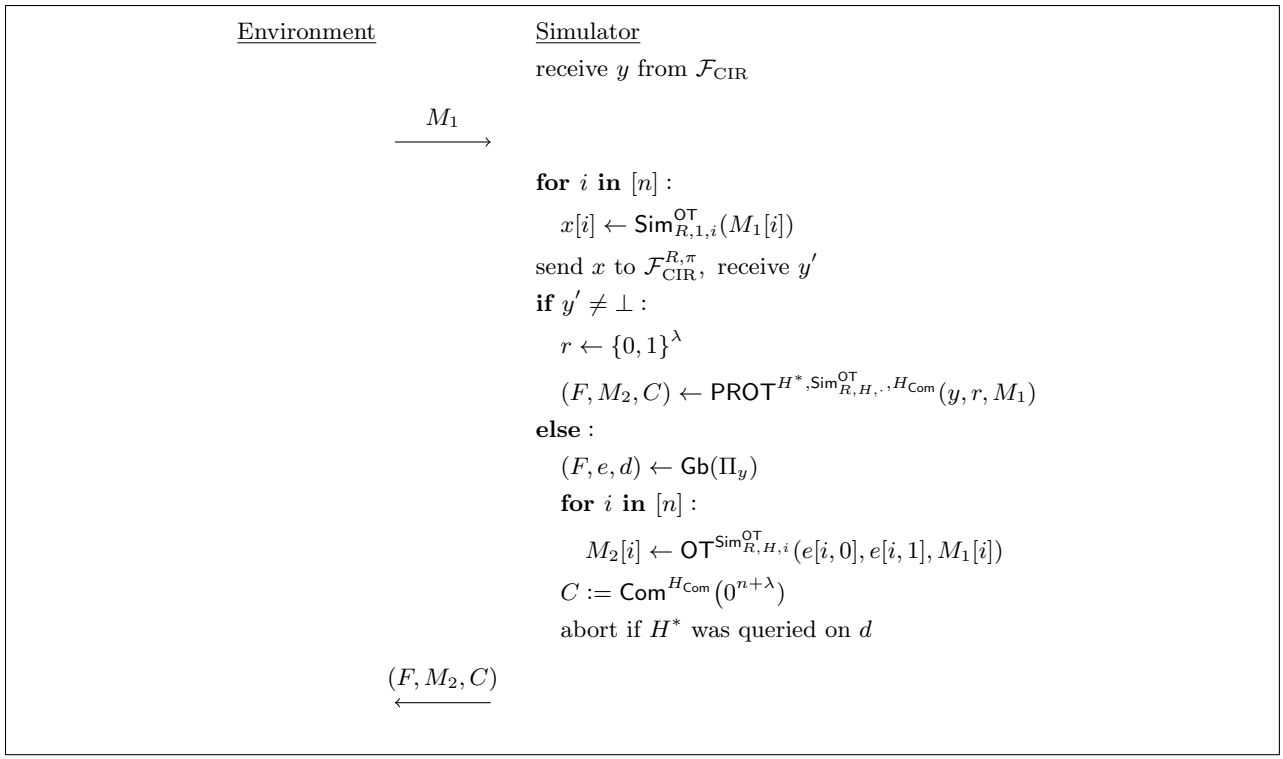

Figure 19: HYBRID$_4$ in the security proof of the CIR-2PC protocol, for a corrupt receiver. We let the garbling and OT protocols sample the randomness implicitly.

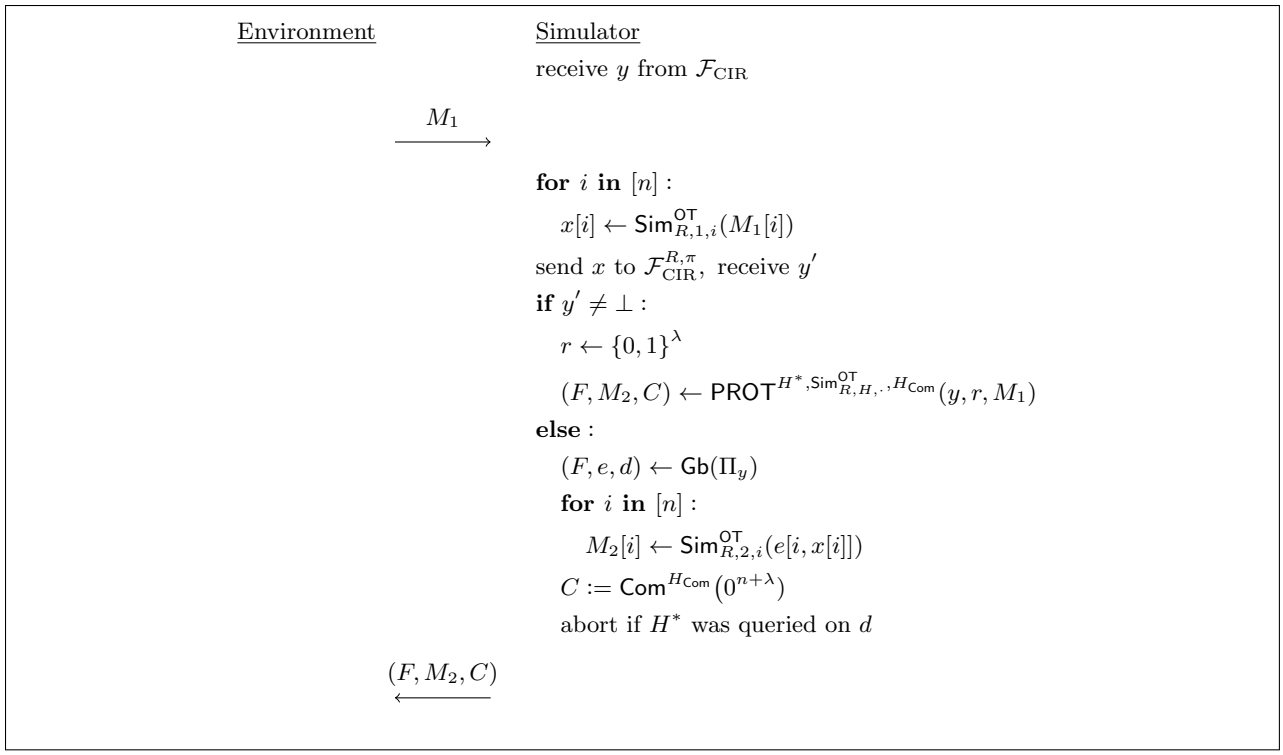

Figure 20: HYBRID$_5$ in the security proof of the CIR-2PC protocol, for a corrupt receiver.

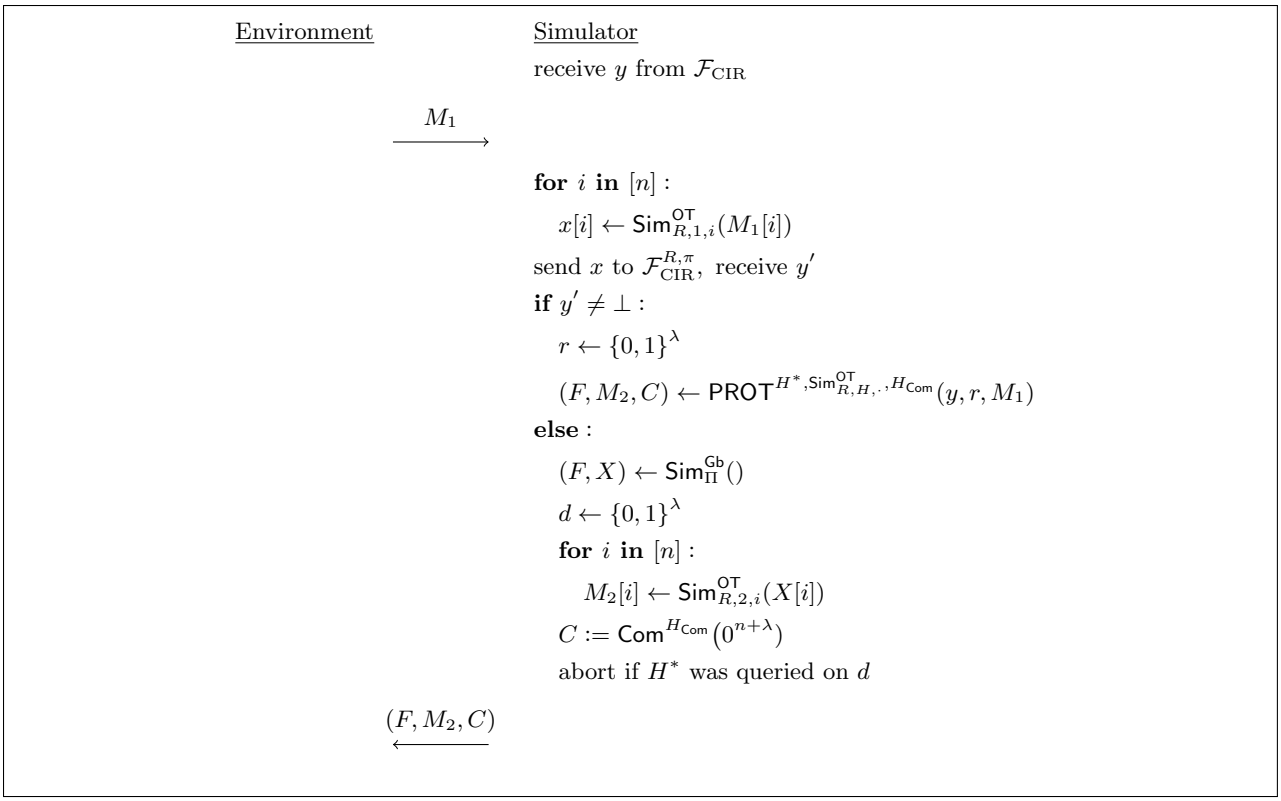

Figure 21: HYBRID_6 in the security proof of the CIR-2PC protocol, for a corrupt receiver.

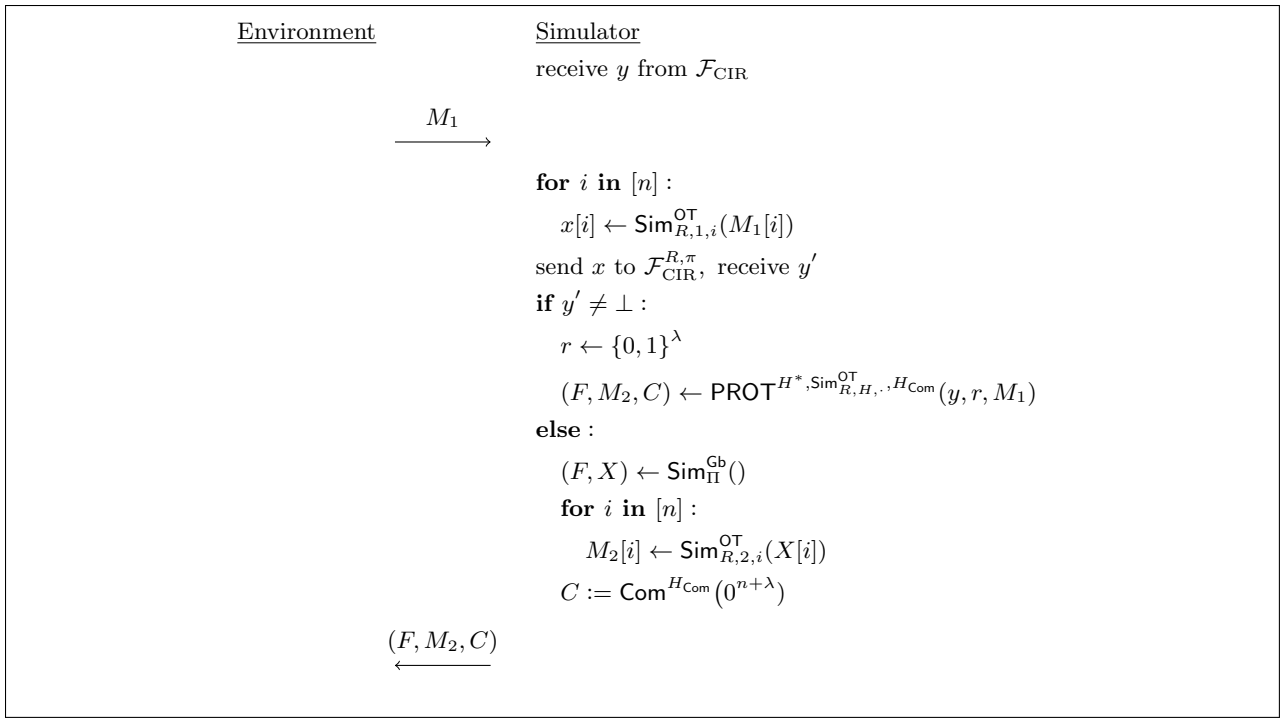

Figure 22: HYBRID$_7$ in the security proof of the CIR-2PC protocol, for a corrupt receiver.

C.3 FPAKE Hybrids

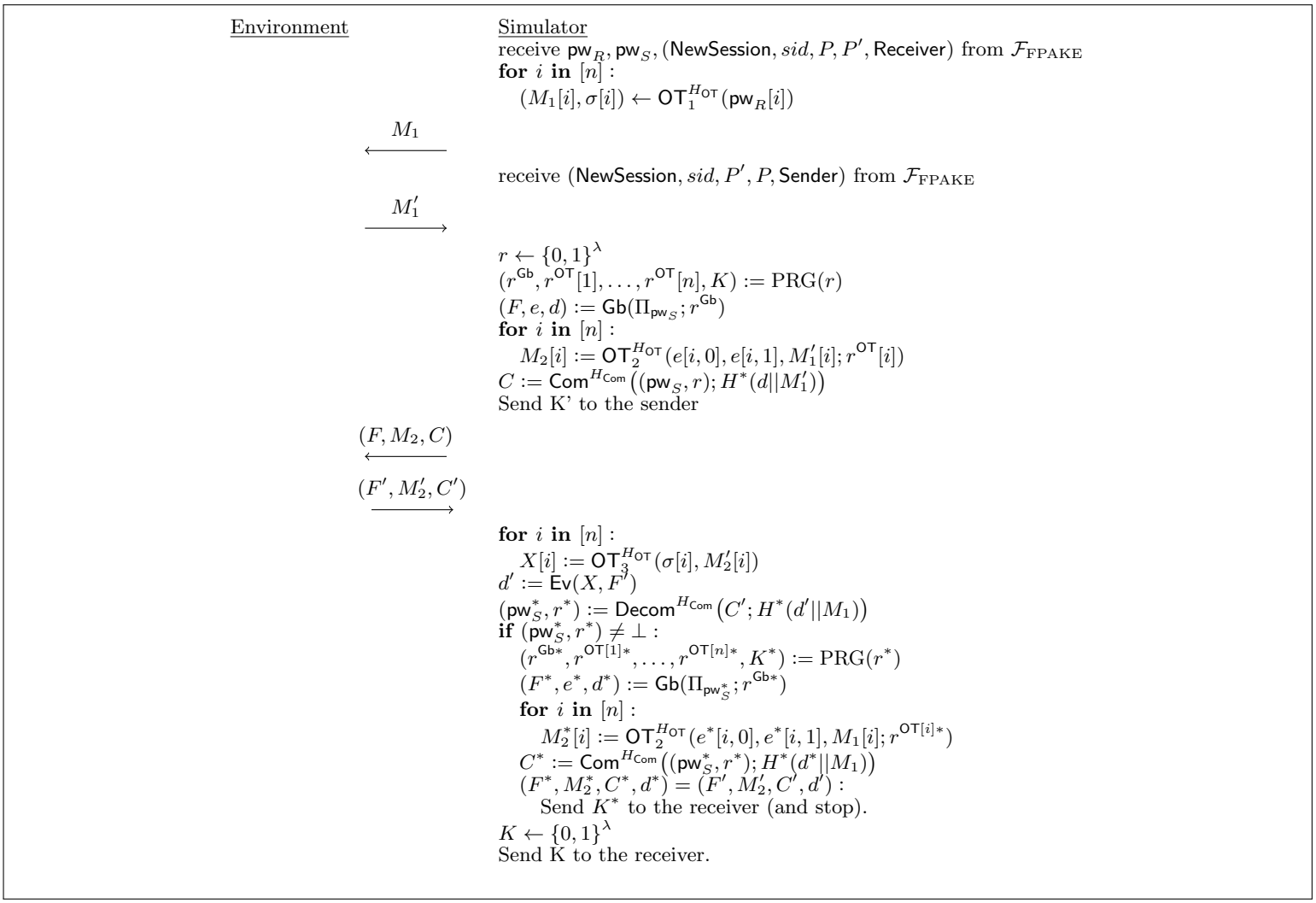

Figure 23: HYBRID_0 in the security proof of the FPAKE protocol.

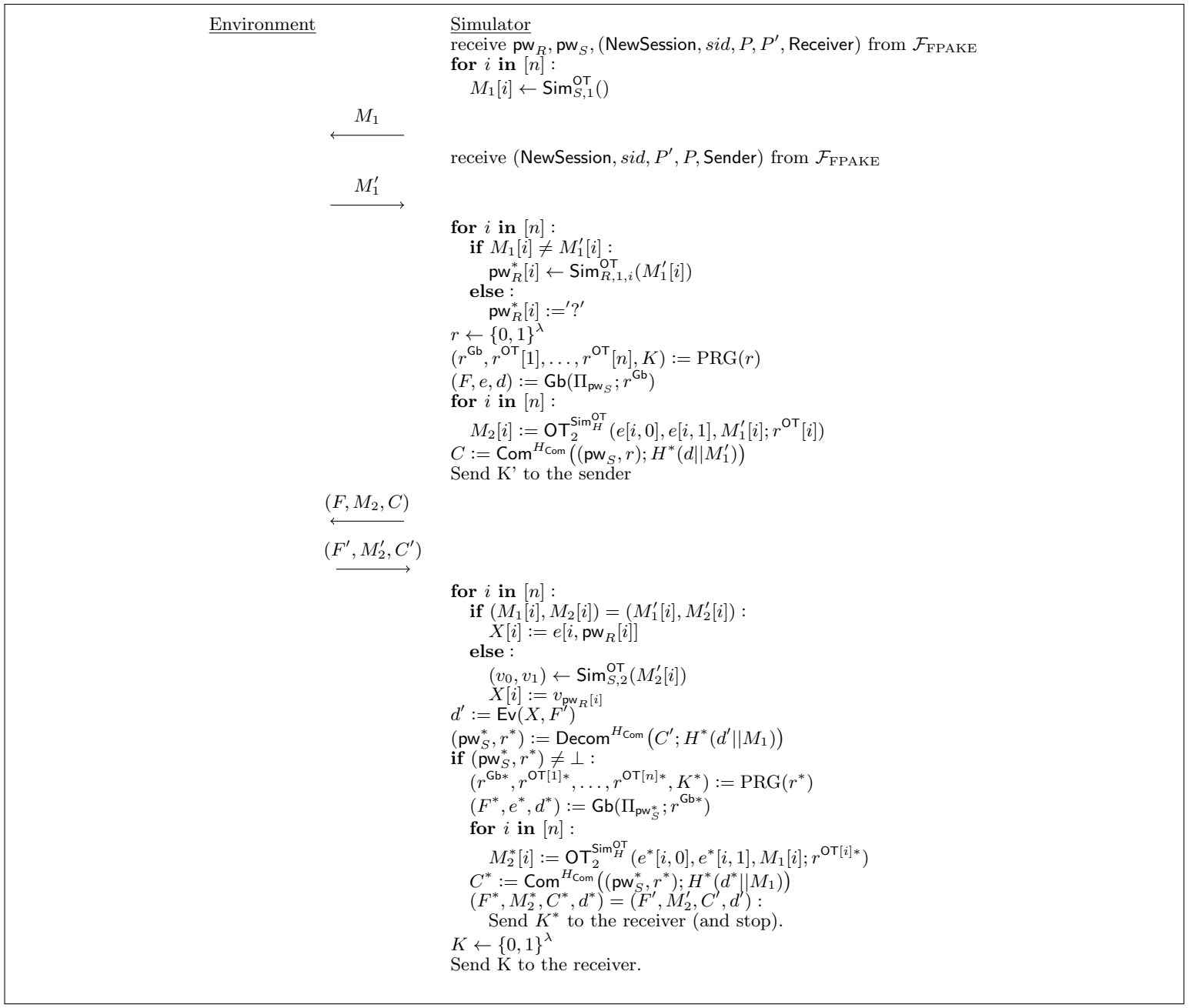

Figure 24: HYBRID$_1$ in the security proof of the FPAKE protocol.



Figure 25: HYBRID$_2$ in the security proof of the FPAKE protocol.

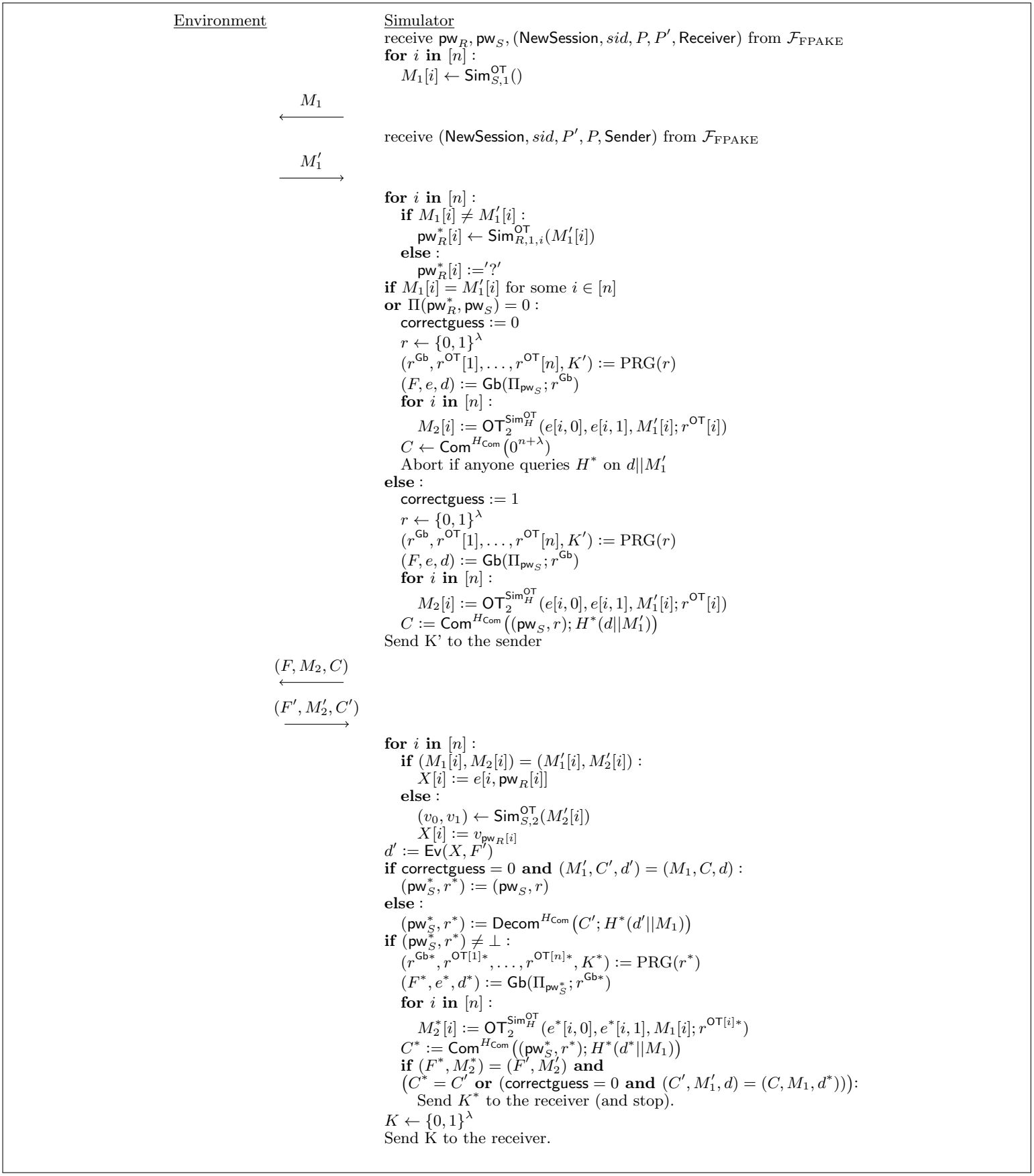

Figure 26: HYBRID$_3$ in the security proof of the FPAKE protocol.

Environment

Simulator

receive $\mathsf{pw}_R, \mathsf{pw}_S, (\mathsf{NewSession}, sid, P, P', \mathsf{Receiver})$ from $\mathcal{F}_{\text{FPAKE}}$

for i **in** $[n]$:

$M_1[i] \leftarrow \mathsf{Sim}^{\mathsf{OT}}_{S,1}()$

$\xleftarrow{M_1}$

receive $(\mathsf{NewSession}, sid, P', P, \mathsf{Sender})$ from $\mathcal{F}_{\text{FPAKE}}$

$\xrightarrow{M_1'}$

for i **in** $[n]$:

if $M_1[i] \neq M_1'[i]$:

$\mathsf{pw}^*_R[i] \leftarrow \mathsf{Sim}^{\mathsf{OT}}_{R,1,i}(M_1'[i])$

else :

$\mathsf{pw}^*_R[i] :=' ?'$

if $M_1[i] = M_1'[i]$ for some $i \in [n]$

or $\Pi(\mathsf{pw}^*_R, \mathsf{pw}_S) = 0$:

$\mathsf{correctguess} := 0$

$r \leftarrow \{0,1\}^\lambda$

$(r^{\mathsf{Gb}}, r^{\mathsf{OT}}[1], \ldots, r^{\mathsf{OT}}[n], K') := \text{PRG}(r)$

$(F, e, d) := \mathsf{Gb}(\Pi_{\mathsf{pw}_S}; r^{\mathsf{Gb}})$

for i **in** $[n]$:

$M_2[i] := \mathsf{OT}_2^{\mathsf{Sim}^{\mathsf{OT}}_H}(e[i,0], e[i,1], M_1'[i]; r^{\mathsf{OT}}[i])$

$C \leftarrow \mathsf{Com}^{H_{\mathsf{Com}}}(0^{n+\lambda})$

Abort if anyone queries H^* on $d||M_1'$

else :

$\mathsf{correctguess} := 1$

$r \leftarrow \{0,1\}^\lambda$

$(r^{\mathsf{Gb}}, r^{\mathsf{OT}}[1], \ldots, r^{\mathsf{OT}}[n], K') := \text{PRG}(r)$

$(F, e, d) := \mathsf{Gb}(\Pi_{\mathsf{pw}_S}; r^{\mathsf{Gb}})$

for i **in** $[n]$:

$M_2[i] := \mathsf{OT}_2^{\mathsf{Sim}^{\mathsf{OT}}_H}(e[i,0], e[i,1], M_1'[i]; r^{\mathsf{OT}}[i])$

$C := \mathsf{Com}^{H_{\mathsf{Com}}}((\mathsf{pw}_S, r); H^*(d||M_1'))$

Send K' to the sender

$\xleftarrow{(F, M_2, C)}$

$\xrightarrow{(F', M_2', C')}$

for i **in** $[n]$:

if $(M_1[i], M_2[i]) = (M_1'[i], M_2'[i])$:

$X[i] := e[i, \mathsf{pw}_R[i]]$

else :

$(v_0, v_1) \leftarrow \mathsf{Sim}^{\mathsf{OT}}_{S,2}(M_2'[i])$

$X[i] := v_{\mathsf{pw}_R[i]}$

$d' := \mathsf{Ev}(X, F')$

if $\mathsf{correctguess} = 0$ **and** $(C', M_1', d') = (C, M_1, d)$:

if $(M_2, F) = (M_2', F')$:

Send K' to the receiver (and stop).

else :

$K \leftarrow \{0,1\}^\lambda$

Send K to the receiver (and stop).

$(\mathsf{pw}^*_S, r^*) := \mathsf{Decom}^{H_{\mathsf{Com}}}(C'; H^*(d'||M_1))$

if $(\mathsf{pw}^*_S, r^*) \neq \perp$:

$(r^{\mathsf{Gb}*}, r^{\mathsf{OT}[1]*}, \ldots, r^{\mathsf{OT}[n]*}, K^*) := \text{PRG}(r^*)$

$(F^*, e^*, d^*) := \mathsf{Gb}(\Pi_{\mathsf{pw}^*_S}; r^{\mathsf{Gb}*})$

for i **in** $[n]$:

$M_2^*[i] := \mathsf{OT}_2^{\mathsf{Sim}^{\mathsf{OT}}_H}(e^*[i,0], e^*[i,1], M_1[i]; r^{\mathsf{OT}[i]*})$

$C^* := \mathsf{Com}^{H_{\mathsf{Com}}}((\mathsf{pw}^*_S, r^*); H^*(d^*||M_1))$

$(F^*, M_2^*, C^*, d^*) = (F', M_2', C', d')$:

Send K^* to the receiver (and stop).

$K \leftarrow \{0,1\}^\lambda$

Send K to the receiver.

Figure 27: HYBRID$_4$ in the security proof of the FPAKE protocol.

Environment		Simulator
		receive $\mathsf{pw}_R, \mathsf{pw}_S, (\mathsf{NewSession}, sid, P, P', \mathsf{Receiver})$ from $\mathcal{F}_{\text{FPAKE}}$
		for i **in** $[n]$:
		$\quad M_1[i] \leftarrow \mathsf{Sim}^{\mathsf{OT}}_{S,1}()$
	$\xleftarrow{M_1}$	
		receive $(\mathsf{NewSession}, sid, P', P, \mathsf{Sender})$ from $\mathcal{F}_{\text{FPAKE}}$
	$\xrightarrow{M_1'}$	
		for i **in** $[n]$:
		$\quad$ **if** $M_1[i] \neq M_1'[i]$:
		$\quad\quad \mathsf{pw}_R^*[i] \leftarrow \mathsf{Sim}^{\mathsf{OT}}_{R,1,i}(M_1'[i])$
		$\quad$ **else** :
		$\quad\quad \mathsf{pw}_R^*[i] :=' ?'$
		if $M_1[i] = M_1'[i]$ for some $i \in [n]$
		or $\Pi(\mathsf{pw}_R^*, \mathsf{pw}_S) = 0$:
		$\quad \mathsf{correctguess} := 0$
		$\quad r \leftarrow \{0,1\}^\lambda$
		$\quad (r^{\mathsf{Gb}}, r^{\mathsf{OT}}[1], \ldots, r^{\mathsf{OT}}[n], K') := \text{PRG}(r)$
		$\quad (F, e, d) := \mathsf{Gb}(\Pi_{\mathsf{pw}_S}; r^{\mathsf{Gb}})$
		$\quad$ **for** i **in** $[n]$:
		$\quad\quad M_2[i] := \mathsf{OT}_2^{\mathsf{Sim}_H^{\mathsf{OT}}}(e[i,0], e[i,1], M_1'[i]; r^{\mathsf{OT}}[i])$
		$\quad C \leftarrow \mathsf{Com}^{H_{\mathsf{Com}}}(0^{n+\lambda})$
		$\quad$ Abort if anyone queries H^* on $d \| M_1'$
		else :
		$\quad \mathsf{correctguess} := 1$
		$\quad r \leftarrow \{0,1\}^\lambda$
		$\quad (r^{\mathsf{Gb}}, r^{\mathsf{OT}}[1], \ldots, r^{\mathsf{OT}}[n], K') := \text{PRG}(r)$
		$\quad (F, e, d) := \mathsf{Gb}(\Pi_{\mathsf{pw}_S}; r^{\mathsf{Gb}})$
		$\quad$ **for** i **in** $[n]$:
		$\quad\quad M_2[i] := \mathsf{OT}_2^{\mathsf{Sim}_H^{\mathsf{OT}}}(e[i,0], e[i,1], M_1'[i]; r^{\mathsf{OT}}[i])$
		$\quad C := \mathsf{Com}^{H_{\mathsf{Com}}}((\mathsf{pw}_S, r); H^*(d \| M_1'))$
		Send K' to the sender
	$\xleftarrow{(F, M_2, C)}$	
	$\xrightarrow{(F', M_2', C')}$	
		for i **in** $[n]$:
		$\quad$ **if** $(M_1[i], M_2[i]) = (M_1'[i], M_2'[i])$:
		$\quad\quad X[i] := e[i, \mathsf{pw}_R[i]]$
		$\quad$ **else** :
		$\quad\quad (v_0, v_1) \leftarrow \mathsf{Sim}^{\mathsf{OT}}_{S,2}(M_2'[i])$
		$\quad\quad X[i] := v_{\mathsf{pw}_R[i]}$
		$d' := \mathsf{Ev}(X, F')$
		if $\mathsf{correctguess} = 0$ **and** $C' = C$:
		$\quad$ **if** $(M_1, M_2, F, d) = (M_1', M_2', F', d')$:
		$\quad\quad$ Send K' to the receiver (and stop).
		$\quad$ **else** :
		$\quad\quad K \leftarrow \{0,1\}^\lambda$
		$\quad\quad$ Send K to the receiver (and stop).
		$(\mathsf{pw}_S^*, r^*) := \mathsf{Decom}^{H_{\mathsf{Com}}}(C'; H^*(d' \| M_1))$
		if $(\mathsf{pw}_S^*, r^*) \neq \perp$:
		$\quad (r^{\mathsf{Gb}*}, r^{\mathsf{OT}[1]*}, \ldots, r^{\mathsf{OT}[n]*}, K^*) := \text{PRG}(r^*)$
		$\quad (F^*, e^*, d^*) := \mathsf{Gb}(\Pi_{\mathsf{pw}_S^*}; r^{\mathsf{Gb}*})$
		$\quad$ **for** i **in** $[n]$:
		$\quad\quad M_2^*[i] := \mathsf{OT}_2^{\mathsf{Sim}_H^{\mathsf{OT}}}(e^*[i,0], e^*[i,1], M_1[i]; r^{\mathsf{OT}[i]*})$
		$\quad C^* := \mathsf{Com}^{H_{\mathsf{Com}}}((\mathsf{pw}_S^*, r^*); H^*(d^* \| M_1))$
		$\quad (F^*, M_2^*, C^*, d^*) = (F', M_2', C', d')$:
		$\quad\quad$ Send K^* to the receiver (and stop).
		$K \leftarrow \{0,1\}^\lambda$
		Send K to the receiver.

Figure 28: HYBRID$_5$ in the security proof of the FPAKE protocol.

Environment

Simulator

receive $\mathsf{pw}_R, \mathsf{pw}_S, (\mathsf{NewSession}, sid, P, P', \mathsf{Receiver})$ from $\mathcal{F}_{\text{FPAKE}}$

for i **in** $[n]$:

$M_1[i] \leftarrow \mathsf{Sim}^{\mathsf{OT}}_{S,1}()$

M_1 $\longleftarrow$

receive $(\mathsf{NewSession}, sid, P', P, \mathsf{Sender})$ from $\mathcal{F}_{\text{FPAKE}}$

M_1' $\longrightarrow$

for i **in** $[n]$:

if $M_1[i] \neq M_1'[i]$:

$\mathsf{pw}^*_R[i] \leftarrow \mathsf{Sim}^{\mathsf{OT}}_{R,1,i}(M_1'[i])$

else :

$\mathsf{pw}^*_R[i] :=' ?'$

if $M_1[i] = M_1'[i]$ for some $i \in [n]$

or $\Pi(\mathsf{pw}^*_R, \mathsf{pw}_S) = 0$:

$\mathsf{correctguess} := 0$

$r \leftarrow \{0,1\}^\lambda$

$(r^{\mathsf{Gb}}, r^{\mathsf{OT}}[1], \ldots, r^{\mathsf{OT}}[n], K') := \text{PRG}(r)$

$(F, e, d) := \mathsf{Gb}(\Pi_{\mathsf{pw}_S}; r^{\mathsf{Gb}})$

for i **in** $[n]$:

$M_2[i] := \mathsf{OT}_2^{\mathsf{Sim}^{\mathsf{OT}}_H}(e[i,0], e[i,1], M_1'[i]; r^{\mathsf{OT}}[i])$

$C \leftarrow \mathsf{Com}^{H_{\mathsf{Com}}}(0^{n+\lambda})$

Abort if anyone queries H^* on $d||M_1'$

else :

$\mathsf{correctguess} := 1$

$r \leftarrow \{0,1\}^\lambda$

$(r^{\mathsf{Gb}}, r^{\mathsf{OT}}[1], \ldots, r^{\mathsf{OT}}[n], K') := \text{PRG}(r)$

$(F, e, d) := \mathsf{Gb}(\Pi_{\mathsf{pw}_S}; r^{\mathsf{Gb}})$

for i **in** $[n]$:

$M_2[i] := \mathsf{OT}_2^{\mathsf{Sim}^{\mathsf{OT}}_H}(e[i,0], e[i,1], M_1'[i]; r^{\mathsf{OT}}[i])$

$C := \mathsf{Com}^{H_{\mathsf{Com}}}((\mathsf{pw}_S, r); H^*(d||M_1'))$

Send K' to the sender

(F, M_2, C) $\longleftarrow$

(F', M_2', C') $\longrightarrow$

for i **in** $[n]$:

if $(M_1[i], M_2[i]) = (M_1'[i], M_2'[i])$:

$X[i] := e[i, \mathsf{pw}_R[i]]$

else :

$(v_0, v_1) \leftarrow \mathsf{Sim}^{\mathsf{OT}}_{S,2}(M_2'[i])$

$X[i] := v_{\mathsf{pw}_R[i]}$

$d' := \mathsf{Ev}(X, F')$

if $\mathsf{correctguess} = 0$ **and** $C' = C$:

if $(M_1, M_2, F) = (M_1', M_2', F')$ **and** $\Pi(\mathsf{pw}_R, \mathsf{pw}_S) = 1$:

Send K' to the receiver (and stop).

else :

$K \leftarrow \{0,1\}^\lambda$

Send K to the receiver (and stop).

$(\mathsf{pw}^*_S, r^*) := \mathsf{Decom}^{H_{\mathsf{Com}}}(C'; H^*(d'||M_1))$

if $(\mathsf{pw}^*_S, r^*) \neq \perp$:

$(r^{\mathsf{Gb}*}, r^{\mathsf{OT}[1]*}, \ldots, r^{\mathsf{OT}[n]*}, K^*) := \text{PRG}(r^*)$

$(F^*, e^*, d^*) := \mathsf{Gb}(\Pi_{\mathsf{pw}^*_S}; r^{\mathsf{Gb}*})$

for i **in** $[n]$:

$M_2^*[i] := \mathsf{OT}_2^{\mathsf{Sim}^{\mathsf{OT}}_H}(e^*[i,0], e^*[i,1], M_1[i]; r^{\mathsf{OT}[i]*})$

$C^* := \mathsf{Com}^{H_{\mathsf{Com}}}((\mathsf{pw}^*_S, r^*); H^*(d^*||M_1))$

$(F^*, M_2^*, C^*, d^*) = (F', M_2', C', d')$:

Send K^* to the receiver (and stop).

$K \leftarrow \{0,1\}^\lambda$

Send K to the receiver.

Figure 29: HYBRID$_6$ in the security proof of the FPAKE protocol.

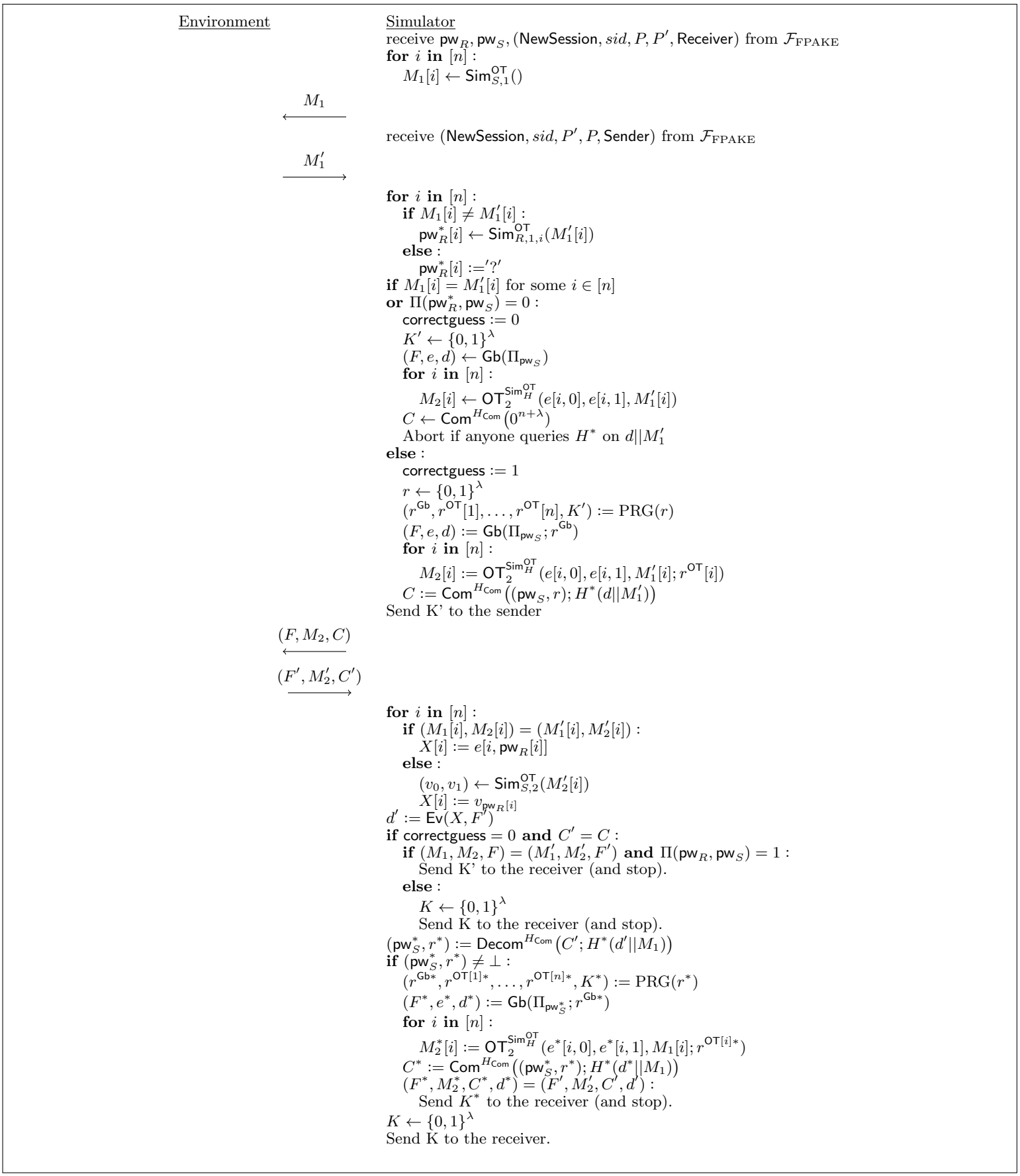

Figure 30: HYBRID$_7$ in the security proof of the FPAKE protocol.

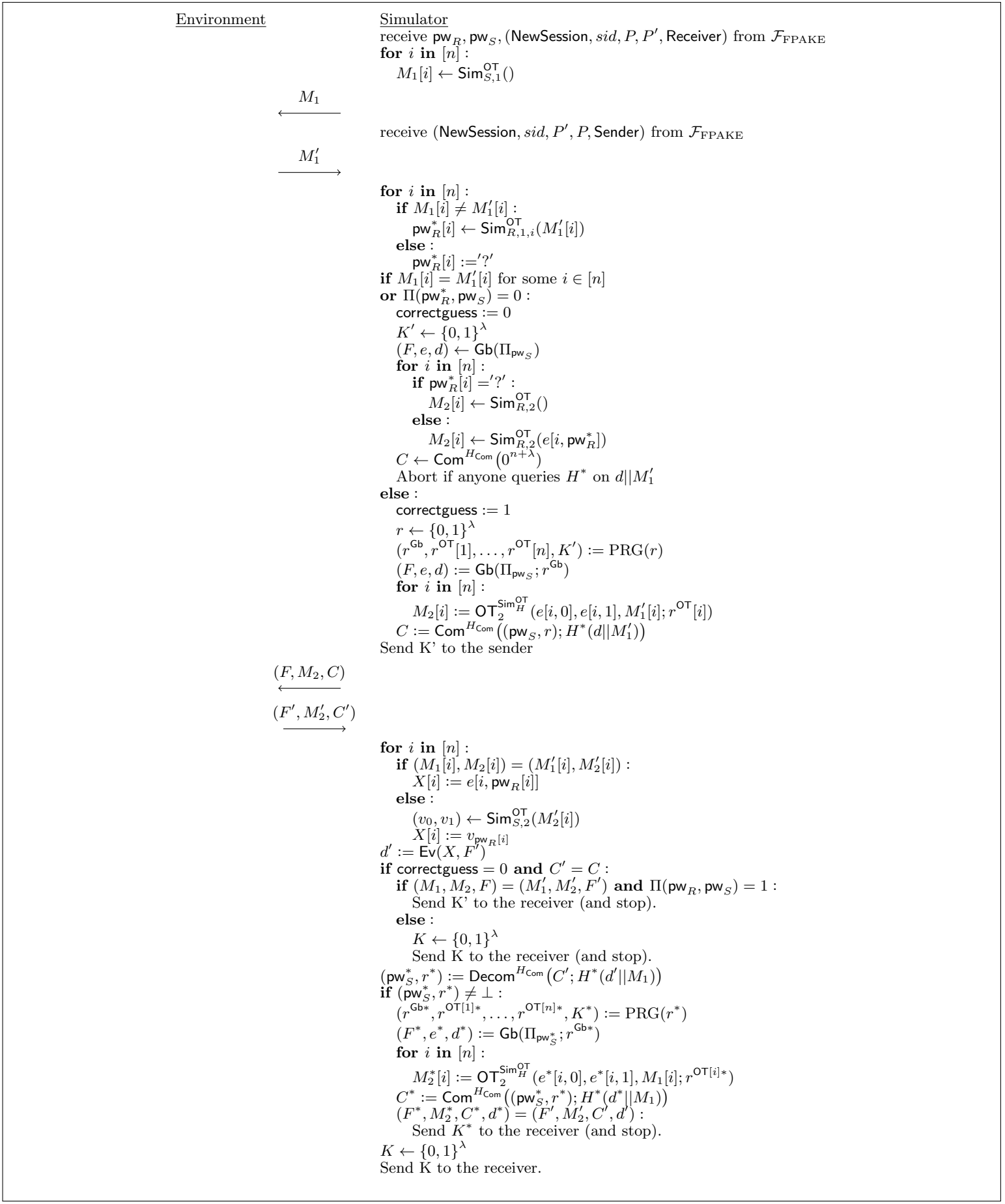

Figure 31: HYBRID$_8$ in the security proof of the FPAKE protocol.

Environment		Simulator
		receive $\mathsf{pw}_R, \mathsf{pw}_S, (\mathsf{NewSession}, sid, P, P', \mathsf{Receiver})$ from $\mathcal{F}_{\text{FPAKE}}$
		for i **in** $[n]$:
		$M_1[i] \leftarrow \mathsf{Sim}^{\mathsf{OT}}_{S,1}()$
	$\xleftarrow{M_1}$	
		receive $(\mathsf{NewSession}, sid, P', P, \mathsf{Sender})$ from $\mathcal{F}_{\text{FPAKE}}$
	$\xrightarrow{M_1'}$	
		for i **in** $[n]$:
		if $M_1[i] \neq M_1'[i]$:
		$\mathsf{pw}^*_R[i] \leftarrow \mathsf{Sim}^{\mathsf{OT}}_{R,1,i}(M_1'[i])$
		else :
		$\mathsf{pw}^*_R[i] :=' ?'$
		if $M_1[i] = M_1'[i]$ for some $i \in [n]$
		or $\Pi(\mathsf{pw}^*_R, \mathsf{pw}_S) = 0$:
		$\mathsf{correctguess} := 0$
		$K' \leftarrow \{0,1\}^\lambda$
		$(F, e, d) \leftarrow \mathsf{Gb}(\Pi_{\mathsf{pw}_S})$
		$X := \mathsf{En}(e, \mathsf{pw}^*_R)$
		for i **in** $[n]$:
		if $\mathsf{pw}^*_R[i] =' ?'$:
		$M_2[i] \leftarrow \mathsf{Sim}^{\mathsf{OT}}_{R,2}()$
		else :
		$M_2[i] \leftarrow \mathsf{Sim}^{\mathsf{OT}}_{R,2}(X[i])$
		$C \leftarrow \mathsf{Com}^{H_{\mathsf{Com}}}\big(0^{n+\lambda}\big)$
		else :
		$\mathsf{correctguess} := 1$
		$r \leftarrow \{0,1\}^\lambda$
		$(r^{\mathsf{Gb}}, r^{\mathsf{OT}}[1], \ldots, r^{\mathsf{OT}}[n], K') := \text{PRG}(r)$
		$(F, e, d) := \mathsf{Gb}(\Pi_{\mathsf{pw}_S}; r^{\mathsf{Gb}})$
		for i **in** $[n]$:
		$M_2[i] := \mathsf{OT}_2^{\mathsf{Sim}^{\mathsf{OT}}_H}(e[i,0], e[i,1], M_1'[i]; r^{\mathsf{OT}}[i])$
		$C := \mathsf{Com}^{H_{\mathsf{Com}}}\big((\mathsf{pw}_S, r); H^*(d \| M_1')\big)$
		Send K' to the sender
	$\xleftarrow{(F, M_2, C)}$	
	$\xrightarrow{(F', M_2', C')}$	
		for i **in** $[n]$:
		if $(M_1[i], M_2[i]) = (M_1'[i], M_2'[i])$:
		$X[i] := e[i, \mathsf{pw}_R[i]]$
		else :
		$(v_0, v_1) \leftarrow \mathsf{Sim}^{\mathsf{OT}}_{S,2}(M_2'[i])$
		$X[i] := v_{\mathsf{pw}_R[i]}$
		$d' := \mathsf{Ev}(X, F')$
		if $\mathsf{correctguess} = 0$ **and** $C' = C$:
		if $(M_1, M_2, F) = (M_1', M_2', F')$ **and** $\Pi(\mathsf{pw}_R, \mathsf{pw}_S) = 1$:
		Send K' to the receiver (and stop).
		else :
		$K \leftarrow \{0,1\}^\lambda$
		Send K to the receiver (and stop).
		$(\mathsf{pw}^*_S, r^*) := \mathsf{Decom}^{H_{\mathsf{Com}}}\big(C'; H^*(d' \| M_1)\big)$
		if $(\mathsf{pw}^*_S, r^*) \neq \perp$:
		$(r^{\mathsf{Gb}*}, r^{\mathsf{OT}[1]*}, \ldots, r^{\mathsf{OT}[n]*}, K^*) := \text{PRG}(r^*)$
		$(F^*, e^*, d^*) := \mathsf{Gb}(\Pi_{\mathsf{pw}^*_S}; r^{\mathsf{Gb}*})$
		for i **in** $[n]$:
		$M_2^*[i] := \mathsf{OT}_2^{\mathsf{Sim}^{\mathsf{OT}}_H}(e^*[i,0], e^*[i,1], M_1[i]; r^{\mathsf{OT}[i]*})$
		$C^* := \mathsf{Com}^{H_{\mathsf{Com}}}\big((\mathsf{pw}^*_S, r^*); H^*(d^* \| M_1)\big)$
		$(F^*, M_2^*, C^*, d^*) = (F', M_2', C', d')$:
		Send K^* to the receiver (and stop).
		$K \leftarrow \{0,1\}^\lambda$
		Send K to the receiver.

Figure 32: HYBRID$_9$ in the security proof of the FPAKE protocol.

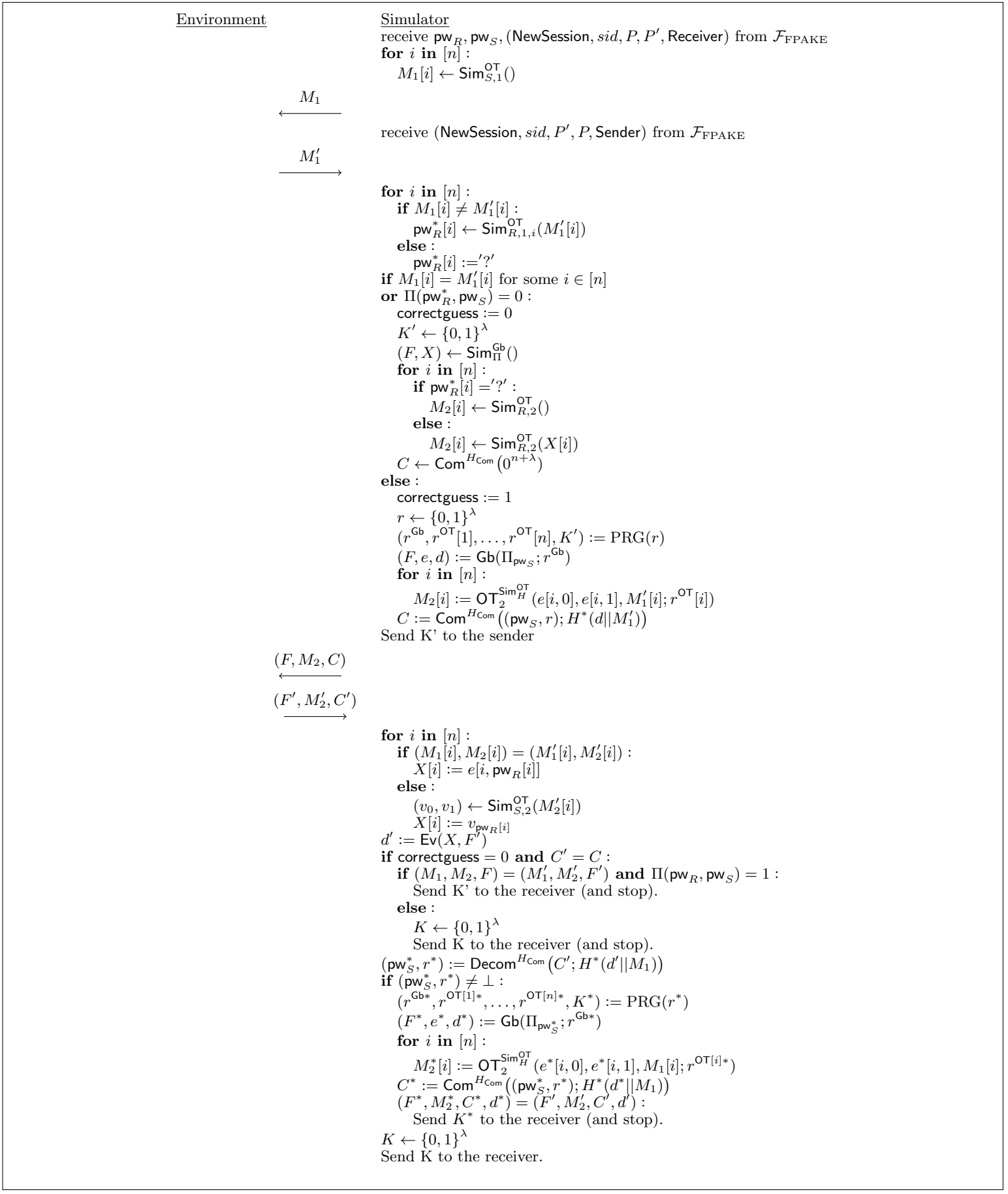

Figure 33: HYBRID$_{10}$ in the security proof of the FPAKE protocol.

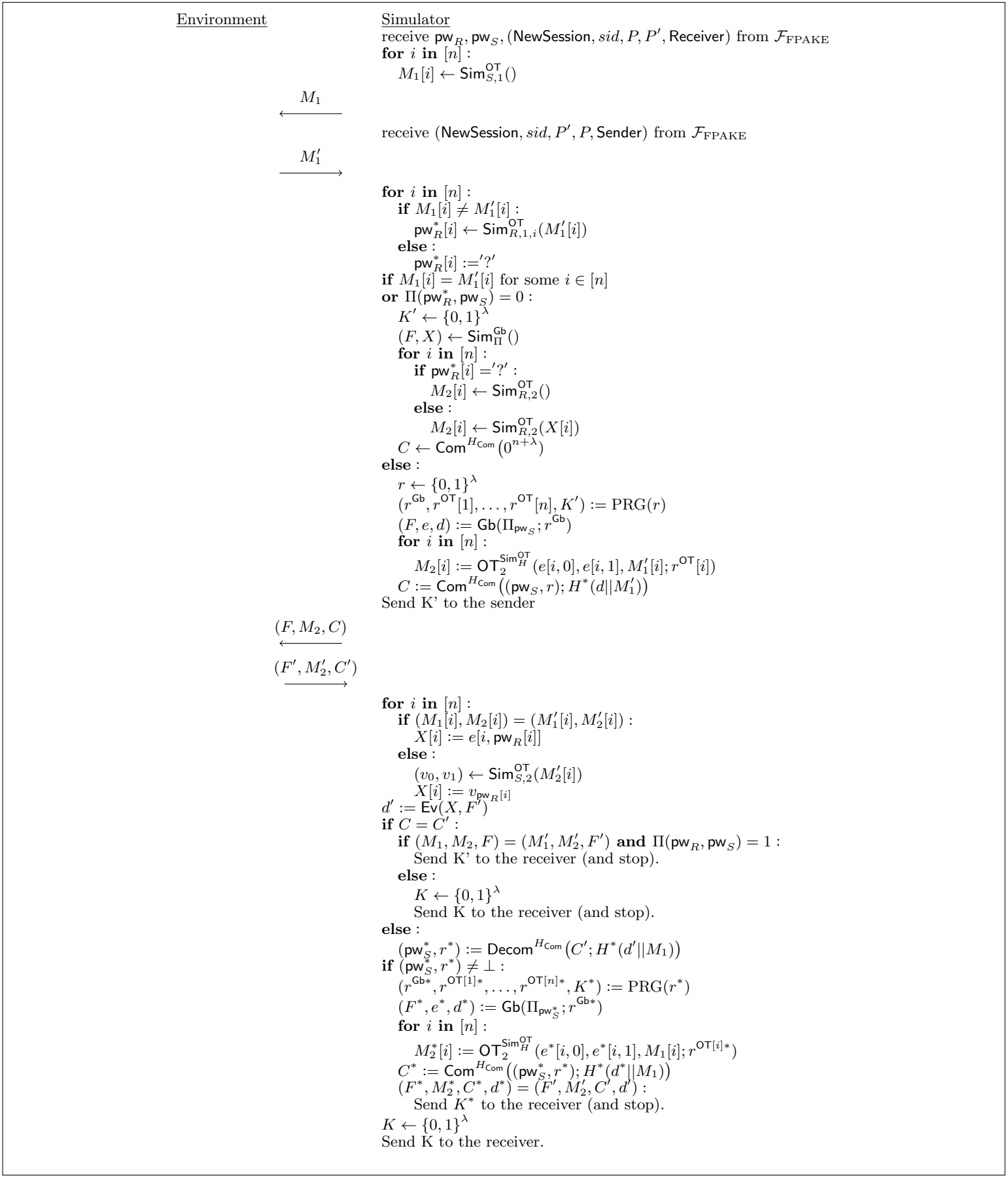

Figure 34: HYBRID$_{11}$ in the security proof of the FPAKE protocol.

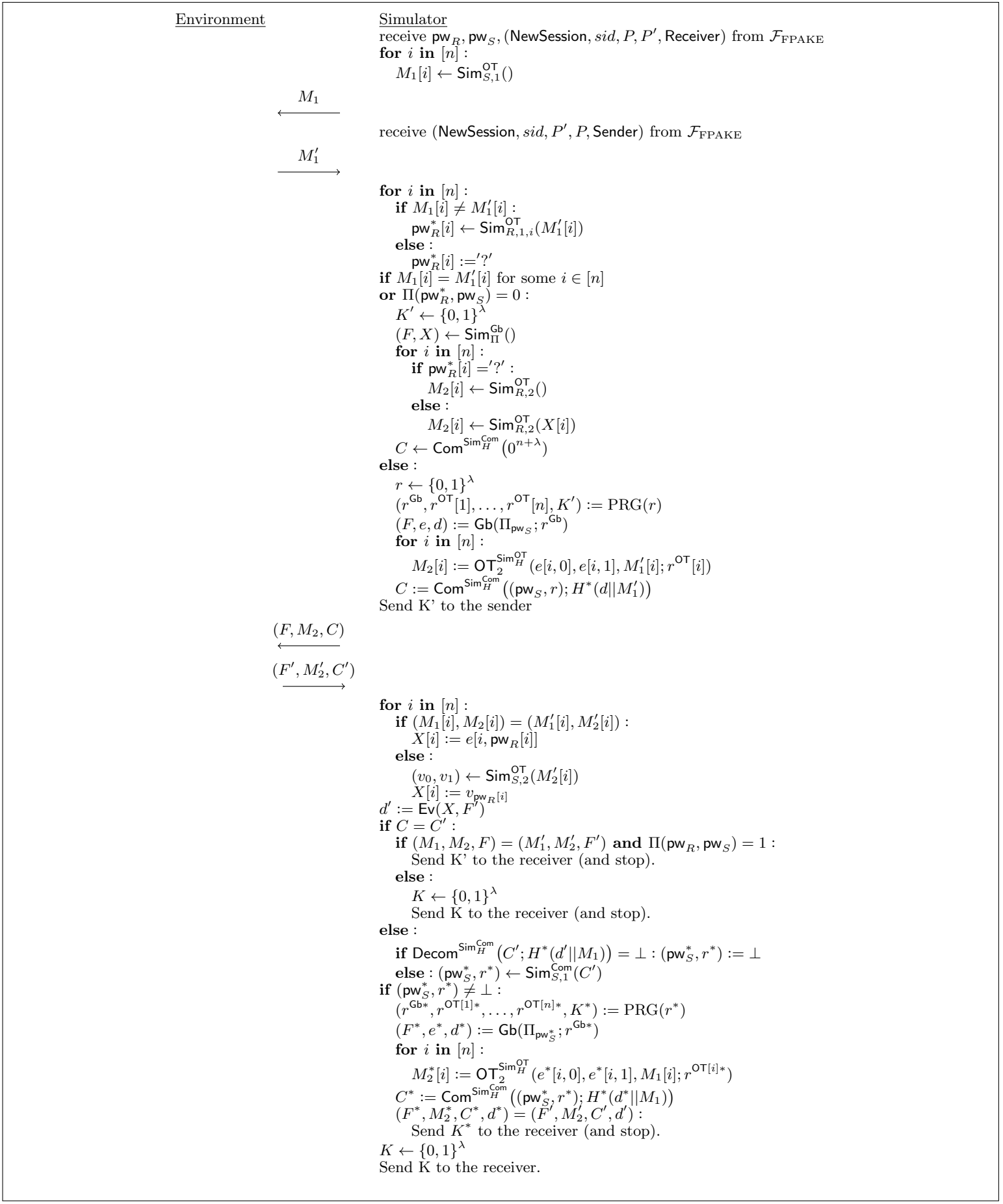

Figure 35: HYBRID_{12} in the security proof of the FPAKE protocol.

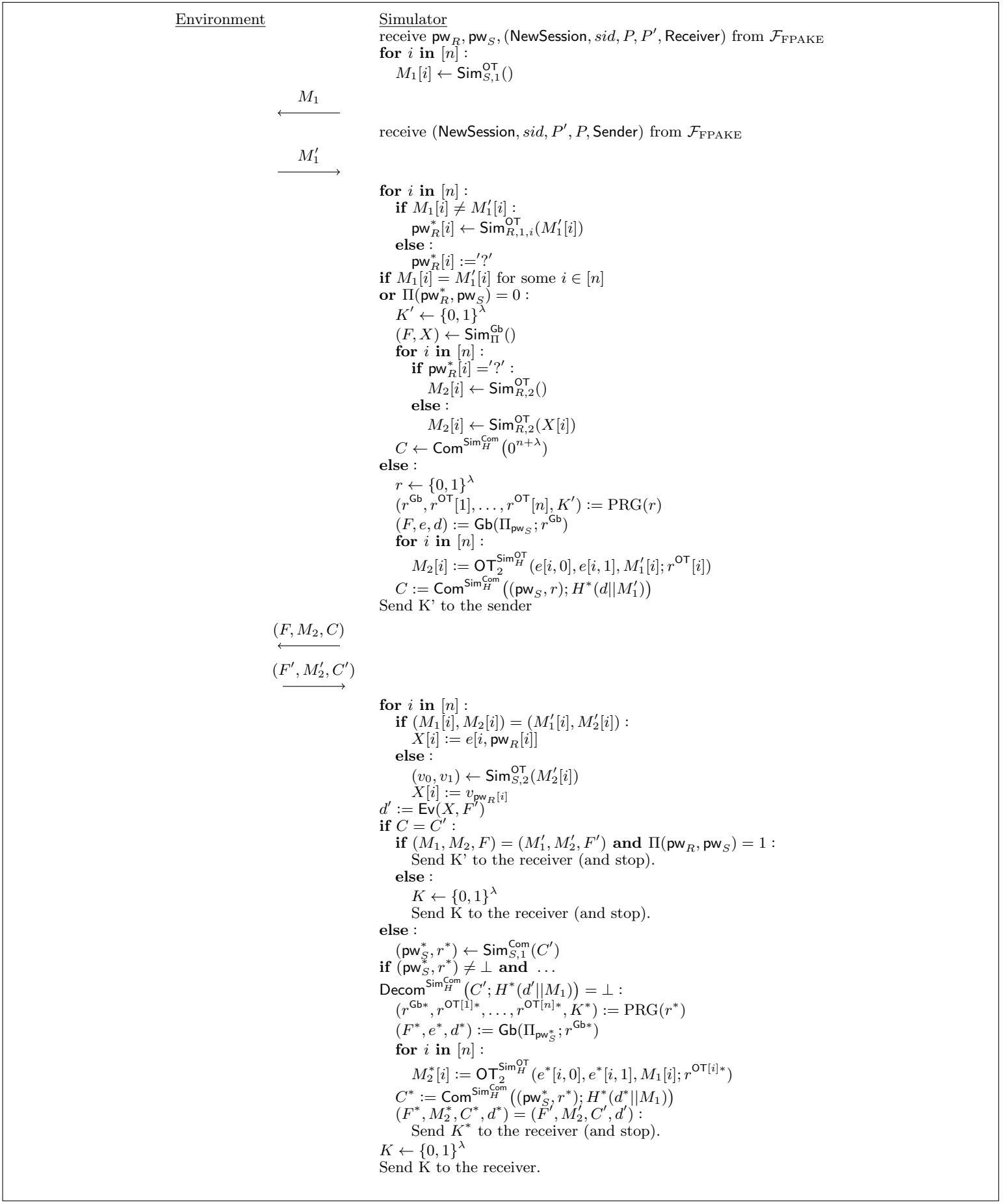

Figure 36: HYBRID$_{13}$ in the security proof of the FPAKE protocol.

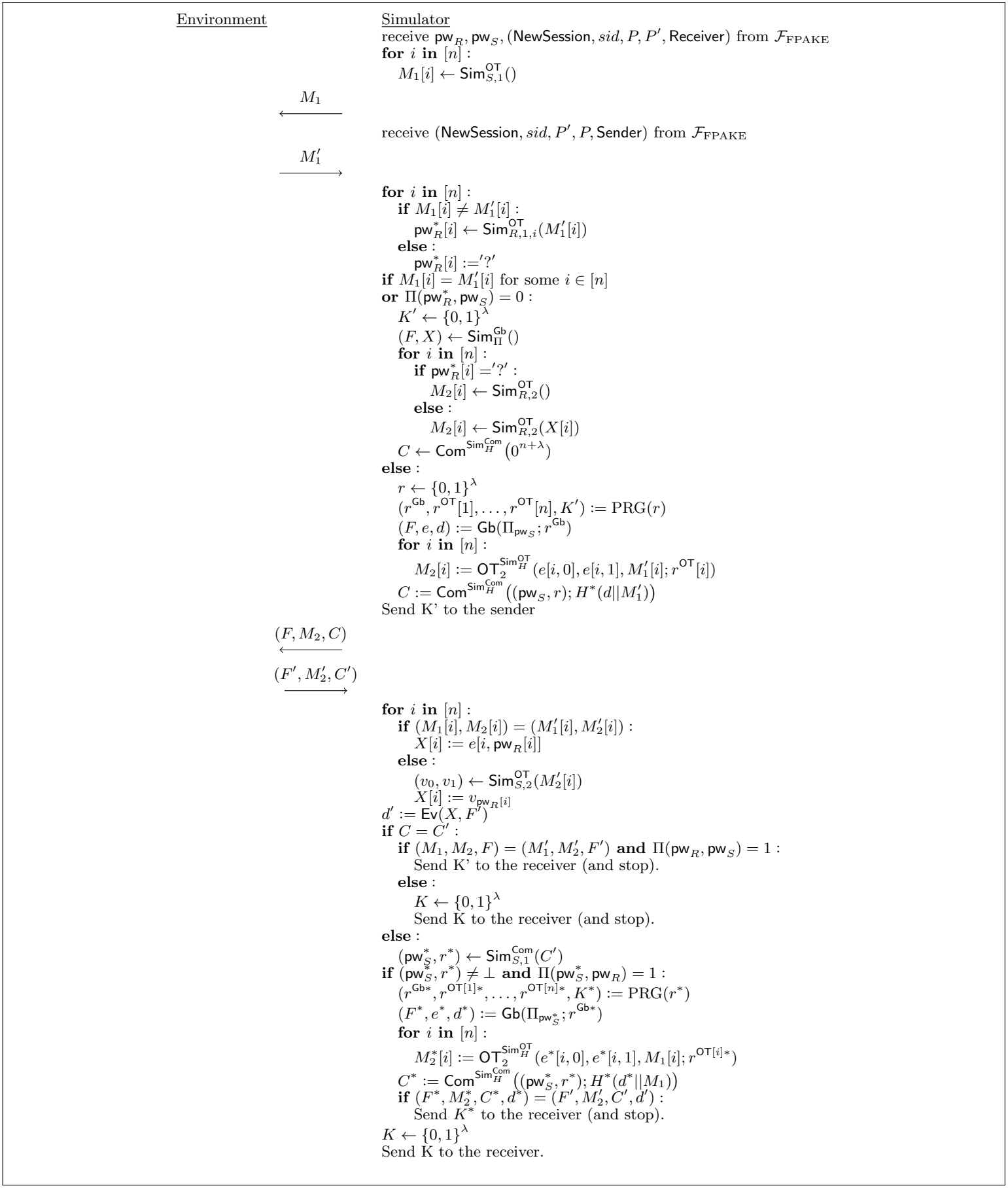

Figure 37: HYBRID$_{14}$ in the security proof of the FPAKE protocol.



Figure 38: HYBRID$_{15}$ in the security proof of the FPAKE protocol.



Figure 39: HYBRID_{16} in the security proof of the FPAKE protocol.

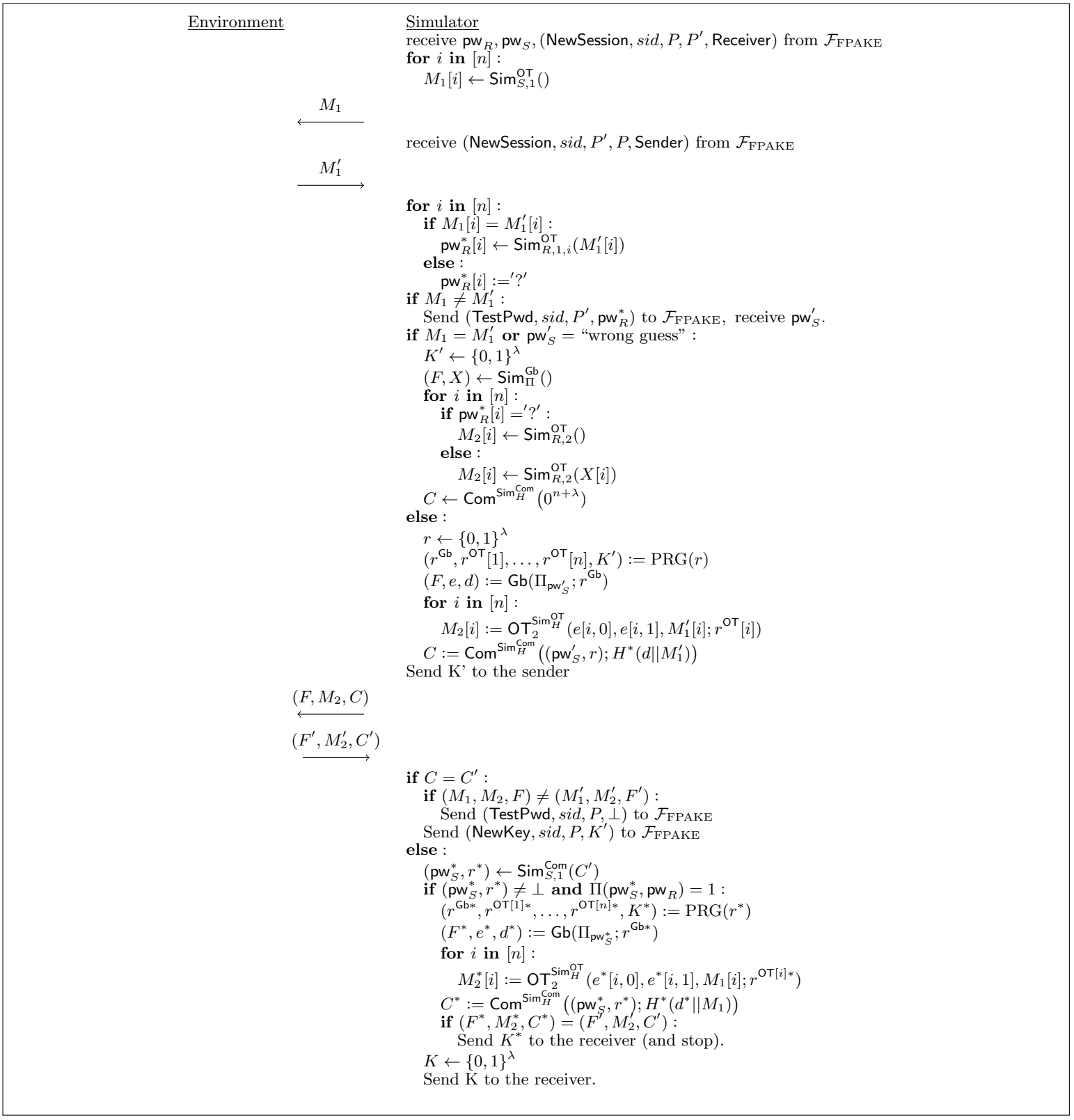

Figure 40: HYBRID$_{17}$ in the security proof of the FPAKE protocol.



Figure 41: HYBRID_{18} in the security proof of the FPAKE protocol.