Optimistic Fair Priced Oblivious Transfer

Alfredo Rial and Bart Preneel

IBBT and Katholieke Universiteit Leuven, ESAT/COSIC
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
`firstname.lastname@esat.kuleuven.be`

Abstract. Priced oblivious transfer (POT) is a two-party protocol between a vendor and a buyer in which the buyer purchases digital goods without the vendor learning what is bought. Although privacy properties are guaranteed, current schemes do not offer fair exchange. A malicious vendor can, e.g., prevent the buyer from retrieving the goods after receiving the payment, and a malicious buyer can also accuse an honest vendor of misbehavior without the vendor being able to prove this untrue. In order to address these problems, we define the concept of optimistic fair priced oblivious transfer and propose a generic construction that extends secure POT schemes to realize this functionality. Our construction, based on verifiably encrypted signatures, employs a neutral adjudicator that is only involved in case of dispute, and shows that disputes can be resolved without the buyer losing her privacy, i.e., the buyer does not need to disclose which digital goods she is interested in. We show that our construction can be instantiated with an existing universally composable POT scheme, and furthermore we propose a novel full-simulation secure POT scheme that is much more efficient.

Keywords: Priced oblivious transfer, verifiably encrypted signatures, fair exchange.

1 Introduction

The protection of privacy in e-commerce is necessary both from a marketing perspective, since privacy concerns discourage buyers from online purchasing [1, 2], as well as from a legal perspective, since different authorities have promulgated regulations to enforce the fulfillment of privacy policies and the confidentiality of buyer's personal data [3]. Additionally, buyers, feeling themselves in an unfavorable position at the payment phase, worry that malicious vendors can, e.g., deliver inappropriate or defective goods later on, and thus e-commerce protocols are normally analyzed in order to prove their fairness [4].

Priced oblivious transfer (POT) is a two-party protocol that provides privacy in e-commerce of digital goods by hiding from the vendor which items are bought. More formally, a vendor $\mathcal{V}$ sells a set of messages $m_1, \dots, m_N$ with prices $p_1, \dots, p_N$ to a buyer $\mathcal{B}$. At each purchase, $\mathcal{B}$ chooses $\tau \in \{1, \dots, N\}$, gets m_τ and pays p_τ without $\mathcal{V}$ learning any information.

Existing POT schemes [5–7] employ a prepaid mechanism where, initially, $\mathcal{B}$ makes a deposit, and, at each purchase, subtracts p_τ from the deposit with $\mathcal{V}$ learning neither p_τ nor the new value of the deposit. Therefore, $\mathcal{B}$ has to trust that $\mathcal{V}$, after receiving the payment, will deliver the requested goods. Furthermore, a malicious $\mathcal{V}$ can claim that $\mathcal{B}$ ran out of funds, and it is impossible for $\mathcal{B}$ to prove this untrue. On the other hand, a malicious $\mathcal{B}$ can claim that $\mathcal{V}$ is not fulfilling his delivery obligations or that her current funds are larger, and again $\mathcal{V}$ cannot prove the contrary. We propose the concept of optimistic fair POT (OFPOT) as a countermeasure.

Previous Work. Fair e-commerce is often seen as a particular case of fair exchange. Early work on fair exchange [8, 9] follows the approach of dividing the items to be exchanged into small pieces and exchanging them piece by piece. As noted in [10], the resulting protocol is unfair in the exchange of the last piece.

More recent work proposes the involvement of a neutral third party [11], which can be split up into several entities to avoid dependence on the reliability of a single entity [12]. When the third party is only involved in case of dispute, the protocol is called optimistic. Some of these protocols are based on verifiably encrypted signatures [13–15]. In a nutshell, $\mathcal{B}$ sends $\mathcal{V}$ a verifiably encrypted signature in her request, and, after $\mathcal{V}$ fulfills his delivery obligations, reveals a valid signature. If $\mathcal{B}$ does not reveal the signature, $\mathcal{V}$ sends the verifiably encrypted signature to the third party, which returns a valid signature after $\mathcal{V}$ demonstrates that he has fulfilled his obligations.

To the best of our knowledge, there were no previous attempts to design a fair e-commerce protocol based on POT. Existing privacy-preserving fair e-commerce protocols [16] provide buyers with anonymity, i.e., they hide from vendors the identity of buyers. As noted in [5, 7], anonymous purchase has some disadvantages, like hindering customer management or making the use of currently deployed online payment methods impossible.

Our Contribution. We define an ideal functionality for OFPOT and we propose a construction that, taking a secure POT scheme as a building block, turns it into an OFPOT scheme. Our definition and construction involve a neutral third party, an adjudicator $\mathcal{A}$, which is only active in case of dispute.

Our construction is based on the use of verifiably encrypted signatures and, to some extent, resembles non-privacy preserving fair e-commerce protocols based on them. Nevertheless, we note that, since in these protocols privacy is not protected, it is trivial for $\mathcal{V}$ to show to $\mathcal{A}$ that he has fulfilled his obligations, and for $\mathcal{A}$ to verify this fact, because $\mathcal{B}$ discloses which item she requests. One of the main contributions of our work is to show that $\mathcal{A}$ can handle complaints from $\mathcal{V}$ and $\mathcal{B}$ by learning neither the list of items offered by $\mathcal{V}$ nor which ones are requested by $\mathcal{B}$.

To be proven full-simulation secure, our construction has to be instantiated with a full-simulation secure POT scheme. We show that the universally composable scheme in [7] is suitable for such an instantiation. However, the schemes in [5, 6] are only half-simulation secure (vendor’s security definition follows the

ideal-world/real-world paradigm, but buyer's security definition is an indistinguishability argument [17]), and thus they are not suitable. Since the scheme in [7] is rather inefficient, we propose a novel and efficient full-simulation secure POT scheme based on the oblivious transfer scheme in [18].

Outline of the Paper. In Section 2 we recall the ideal functionality for POT given in [7] and we describe our novel POT scheme. In Section 3 we define an ideal functionality for OFPOT and we depict our OFPOT scheme. Finally, Section 4 draws a conclusion and discusses future work.

2 Efficient Priced Oblivious Transfer

2.1 Technical Preliminaries

A function ν is *negligible* if, for every integer c , there exists an integer K such that for all $k > K$, $|\nu(k)| < 1/k^c$. A problem is said to be *hard* (or *intractable*) if there exists no probabilistic polynomial time (p.p.t.) algorithm that solves it with non-negligible probability (in the size of the input or the security parameter).

Bilinear maps. Let $\mathbb{G}$ and $\mathbb{G}_t$ be groups of prime order p . A map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_t$ must satisfy bilinearity, i.e., $e(g^x, g^y) = e(g, g)^{xy}$; non-degeneracy, i.e., for all generators $g \in \mathbb{G}$, $e(g, g)$ generates $\mathbb{G}_t$; and efficiency, i.e., there exists an efficient algorithm $\text{BGen}(1^\kappa)$ that outputs the pairing group setup $(p, \mathbb{G}, \mathbb{G}_t, e, g)$ and an efficient algorithm to compute $e(a, b)$ for any $a, b \in \mathbb{G}$.

Security Assumptions. Let $(p, \mathbb{G}, \mathbb{G}_t, e, g)$ be a pairing group setup. The Strong Diffie-Hellman assumption (l -SDH) [19] states that, on input $(g, g^x, \dots, g^{x^l}) \in \mathbb{G}^{l+1}$ for $x \leftarrow \mathbb{Z}_p$, it is hard to output a pair $(c, g^{1/(x+c)})$ for $c \leftarrow \mathbb{Z}_p$. The Power Decisional Diffie-Hellman assumption (l -PDDH) [18] states that, on input $(g, g^x, \dots, g^{x^l}, H) \in \mathbb{G}^{l+1} \times \mathbb{G}_t$ for random H and $x \leftarrow \mathbb{Z}_p$, it is hard to distinguish between the vector $(H^x, H^{x^2}, \dots, H^{x^l}) \in \mathbb{G}_t^l$ and a random vector $T \in \mathbb{G}_t^l$. As shown in [20], the l -PDDH assumption is implied by the l -BDHE assumption [21].

Signature Schemes. A signature scheme consists of the algorithms (Kg , Sign , Vf). $\text{Kg}(1^\kappa)$ outputs a key pair (sk, pk) . $\text{Sign}(sk, m)$ outputs a signature σ on message m . $\text{Vf}(pk, \sigma, m)$ outputs **accept** if σ is a valid signature on m and **reject** otherwise. A signature scheme must be correct and unforgeable [22]. Informally speaking, correctness implies that Vf always accepts a valid signature. Unforgeability means that no p.p.t adversary should be able to output a message-signature pair (σ, m) unless he has previously obtained a signature on m .

We employ the weakly secure signature scheme in [19], which utilizes the SDH assumption. This scheme is existentially unforgeable under a weak chosen message attack, where the adversary submits all signature queries before seeing

the public key. The setup consists of the pairing group setup $(p, \mathbb{G}, \mathbb{G}_t, e, g)$. $\text{Wkg}(1^\kappa)$ picks random $x \leftarrow \mathbb{Z}_p$ and outputs a key pair $(sk, pk) = (x, g^x)$. $\text{WSign}(sk, m)$ computes $\sigma = g^{1/(x+m)}$. $\text{WVf}(pk, \sigma, m)$ outputs **accept** if $e(\sigma, pk \cdot g^m) = e(g, g)$.

Commitment schemes. A non-interactive commitment scheme consists of the algorithms $(\text{ComS}, \text{Com}, \text{Open})$. $\text{ComS}(1^\kappa)$ generates the parameters of the commitment scheme par_{com} . $\text{Com}(par_{com}, x, open)$ outputs a commitment C to x using auxiliary information $open$. A commitment is opened by revealing $(x, open)$ and checking whether $\text{Open}(par_{com}, C, x, open)$ outputs **accept**. A commitment scheme has a hiding property and a binding property. Informally speaking, the hiding property ensures that a commitment C to x does not reveal any information about x , whereas the binding property ensures that C cannot be opened to another value x' . When it is clear from the context, we omit the commitment parameters par_{com} .

We employ the commitment scheme proposed by Pedersen [23]. $\text{PComS}(1^\kappa)$ picks random generators g, h of a group $\mathbb{G}$ of prime order p and outputs $par_{com} = (g, h)$. $\text{PCom}(par_{com}, x, open_x)$ outputs $C = g^x h^{open_x}$ on input $x \in \mathbb{Z}_p$ and randomness $open_x \in \mathbb{Z}_p$. $\text{POpen}(par_{com}, C, x', open'_x)$ outputs **accept** if $C = g^{x'} h^{open'_x}$. This scheme is information theoretically hiding and computationally binding under the discrete logarithm assumption.

Proofs of Knowledge. A zero-knowledge proof of knowledge [24] is a two-party protocol between a prover and a verifier. The prover proves to the verifier knowledge of some secret input that fulfills some statement without disclosing this input to the verifier. The protocol should fulfill two properties. First, it should be a proof of knowledge, i.e., a prover without knowledge of the secret input convinces the verifier with negligible probability. More technically, there exists a knowledge extractor that extracts the secret input from a successful prover with all but negligible probability. Second, it should be zero-knowledge, i.e., the verifier does not learn any information about the secret input. More technically, for all possible verifiers there exists a simulator that, without knowledge of the secret input, yields a distribution that cannot be distinguished from the interaction with a real prover.

We use several existing results to prove statements about discrete logarithms: (1) proof of knowledge of a discrete logarithm modulo a prime [25]; (2) proof of knowledge of the equality of some element in different representations [26]; (3) proof of knowledge that a value α lies in a given interval $[0, A]$ [27]; and (4) proof of the disjunction or conjunction of any two of the previous [28]. These results are often given in the form of Σ -protocols but they can be turned into zero-knowledge protocols using efficient zero-knowledge compilers [29, 30], and they can be turned into non-interactive proofs in the random oracle model via the Fiat-Shamir heuristic [31].

When referring to the proofs above, we follow the notation introduced by Camenisch and Stadler [32] for various proofs of knowledge of discrete log-

arithms and proofs of the validity of statements about discrete logarithms. $\text{NIPK}\{(\alpha, \beta, \delta) : y = g^\alpha h^\beta \wedge \tilde{y} = \tilde{g}^\alpha \tilde{h}^\delta \wedge A \leq \alpha \leq B\}$ denotes a “*non-interactive zero-knowledge proof of knowledge of integers α , β , and δ such that $y = g^\alpha h^\beta$, $\tilde{y} = \tilde{g}^\alpha \tilde{h}^\delta$ and $A \leq \alpha \leq B$ holds*”, where $y, g, h, \tilde{y}, \tilde{g}$, and $\tilde{h}$ are elements of some groups $G = \langle g \rangle = \langle h \rangle$ and $\tilde{G} = \langle \tilde{g} \rangle = \langle \tilde{h} \rangle$ that have the same order. (Note that some elements in the representation of y and $\tilde{y}$ are equal.) The convention is that letters in the parenthesis, in this example α , β , and δ , denote quantities whose knowledge is being proven, while all other values are known to the verifier.

We employ the range proof proposed in [27] to prove that a value α lies in an interval $[0, d^a)$. The proof has an initialization phase where the verifier provides the prover with signatures that allow the prover to write the value α in base- d . First, the verifier runs $\text{InitVer}(1^\kappa, D_{\max})$, which runs $\text{WKg}(1^\kappa)$ to get a key pair (sk, pk) and computes signatures $A_i = \text{WSign}(sk, i)$ on d -ary digits, i.e., $i \in \mathbb{Z}_d$. It also runs $\text{PEComS}(1^\kappa)$ to get par_{com} and sets $par_{ran} = (pk, \{A_i\}_{i \in \mathbb{Z}_d}, par_{com})$. The verifier sends par_{ran} to the prover. The prover runs $\text{InitP}(par_{ran})$ to verify the signatures by running, for $i \in \mathbb{Z}_d$, $\text{WVf}(pk, A_i, i)$. After this initialization phase, prover and verifier can run multiple proofs. The verifier receives as input a commitment C , and the prover values $(\alpha, open_\alpha)$ such that $C = \text{PECom}(par_{com}, \alpha, open_\alpha)$. The prover runs $\text{RProve}(par_{ran}, \alpha, open_\alpha)$, which picks $v_j \leftarrow \mathbb{Z}_p$ and computes $V_j = A_{\alpha_j}^{v_j}$ for every $j \in \mathbb{Z}_a$ such that $\alpha = \sum_{j \in \mathbb{Z}_a} \alpha_j d^j$, and a proof $pok = \text{NIPK}\{(open_\alpha, \{\alpha_j, v_j\}_{j \in \mathbb{Z}_a}) : C = h^{open_\alpha} \prod_{j \in \mathbb{Z}_a} (g^{d^j})^{\alpha_j} \wedge V_j = g^{v_j/(sk+\alpha_j)}\}$. (We abbreviate it as $\text{NIPK}\{(\alpha, open_\alpha) : 0 \leq \alpha < d^a \wedge C = \text{PEComS}(par_{com}, \alpha, open_\alpha)\}$.) The prover sends $(\{V_j\}_{j \in \mathbb{Z}_a}, pok)$ to the verifier.

Public Key Encryption A public key encryption scheme consists of the algorithms $(\text{Kg}, \text{Enc}, \text{Dec})$. $\text{Kg}(1^\kappa)$ outputs a key pair (sk, pk) . $\text{Enc}(pk, m)$ outputs a ciphertext ct on input pk and a message m . $\text{Dec}(sk, ct)$ outputs the message m .

2.2 Definition

We define security following the ideal-world/real-world paradigm [33]. In the real world, a set of parties interact according to the protocol description in the presence of a real adversary $\mathcal{E}$, while in the ideal world dummy parties interact with an ideal functionality that carries out the desired task in the presence of an ideal adversary $\mathcal{S}$. A protocol ψ is secure if there exists no environment $\mathcal{Z}$ that can distinguish whether it is interacting with adversary $\mathcal{E}$ and parties running protocol ψ or with the ideal process for carrying out the desired task, where ideal adversary $\mathcal{S}$ and dummy parties interact with an ideal functionality $\mathcal{F}_\psi$. More formally, we say that protocol ψ emulates the ideal process when, for any adversary $\mathcal{E}$, there exists a simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$, the ensembles $\text{IDEAL}_{\mathcal{F}_\psi, \mathcal{S}, \mathcal{Z}}$ and $\text{REAL}_{\psi, \mathcal{E}, \mathcal{Z}}$ are computationally indistinguishable. We refer to [33] for a description of how these ensembles are constructed.

We recall the ideal functionality $\mathcal{F}_{\text{POT}}$ for priced oblivious transfer in [7]. Every functionality and every protocol invocation should be instantiated with a

unique session-ID that distinguishes it from other instantiations. For the sake of ease of notation, we omit session-IDs from our description.

Functionality $\mathcal{F}_{\text{POT}}$

Parameterized with the number of messages N , the message length l , the maximum price $p_{\max}$, and the maximum deposit $D_{\max}$, and running with a vendor $\mathcal{V}$ and a buyer $\mathcal{B}$, $\mathcal{F}_{\text{POT}}$ works as follows:

- On input a message $(\text{init}, m_1, p_1, \dots, m_N, p_N)$ from $\mathcal{V}$, where each $m_i \in \{0, 1\}^l$ and each $p_i \in [0, p_{\max}]$, it stores $(m_1, p_1, \dots, m_N, p_N)$ and sends $(\text{init}, p_1, \dots, p_N)$ to $\mathcal{B}$.
- On input a message $(\text{deposit}, ac)$, where $ac \in [0, D_{\max}]$, if a $(\text{init}, \dots)$ message was not received before, then it does nothing. Otherwise, it stores ac and sends $(\text{deposit}, ac)$ to $\mathcal{V}$.
- On input a message $(\text{request}, \tau)$ from $\mathcal{B}$, where $\tau \in \{1, \dots, N\}$, if either messages $(\text{init}, m_1, p_1, \dots, m_N, p_N)$ and $(\text{deposit}, ac)$ were not received before or $ac - p_\tau < 0$, then it does nothing. Otherwise, it sends (request) to $\mathcal{V}$ and receives $(\text{response}, b)$ in response. If $b = 0$, it sends $(\text{response}, \perp)$ to $\mathcal{B}$. If $b = 1$, it updates $ac = ac - p_\tau$ and sends $(\text{response}, m_\tau)$ to $\mathcal{B}$.

2.3 Intuition Behind Our Construction

Our POT scheme is based on the adaptive oblivious transfer scheme in [18]. It is an assisted decryption scheme in which there is an initialization phase and several purchase phases. At the initialization phase, $\mathcal{V}$ sends $\mathcal{B}$ a collection of ciphertexts that encrypt the messages $m_1, \dots, m_N$ to be sold, and at each purchase $\mathcal{V}$ helps $\mathcal{B}$ to decrypt one of them. As noted in [34], this design approach leads to purchase phases with constant communication and computation complexity, and ensures that $\mathcal{V}$ cannot modify the messages after the initialization phase.

Each ciphertext C_i consists of a unique price¹ p_i , a signature on the price A_i and an encryption $B_i = e(h, A_i) \cdot m_i$, where h is the secret key of $\mathcal{V}$. To compute a request for m_τ , $\mathcal{B}$ sends $\mathcal{V}$ a blinded value $V = A_\tau^v$ together with a zero-knowledge proof that $\mathcal{B}$ possesses a valid signature A_τ and that V is correctly computed. $\mathcal{V}$ computes a blinded decryption $W = e(h, V)$ and proves that the secret key h was used to compute W . Finally, $\mathcal{B}$ unblinds W to decrypt the message m_τ via $B_\tau / (W^{1/v})$.

To allow for oblivious payments, our POT scheme follows the prepaid mechanism by [7]. At the initialization phase, $\mathcal{B}$ makes a deposit ac_0 and sends a commitment D_0 to the deposit and its opening to $\mathcal{V}$. At purchase phase i , $\mathcal{B}$

¹ If several messages have the same price, unique prices can be obtained by adequately scaling prices and accounts [5].

sends a commitment D_i to the new value of the account ac_i along with a zero-knowledge proof that $ac_i = ac_{i-1} - p_\tau$ and that $ac_i \in [0, D_{max})$, to prove that the account is correctly updated and that $\mathcal{B}$ has enough funds to buy m_τ . For the latter, we employ the range proof recently proposed in [27].

We prove that our scheme realizes $\mathcal{F}_{\text{POT}}$ under the SDH and PDDH assumptions (used in [18]), and under the assumption the binding and hiding properties of the commitment scheme hold. In the description of the scheme given below, we compute non-interactive proofs via Fiat-Shamir heuristic [31] and thus the construction is secure in the random oracle model [35]. This is convenient when using the scheme as a building block of the OFPOT scheme depicted in Section 3, because it allows $\mathcal{A}$ to verify requests and responses non-interactively. Nevertheless, for other uses of the scheme interactive proofs of knowledge can be employed, yielding a construction in the standard model.

2.4 Description of the Scheme

We begin with a high level description of the POT scheme. Details on the algorithms can be found below.

POT

Initialization phase. On input $(\text{init}, m_1, p_1, \dots, m_N, p_N)$, in which each price is unique, $\mathcal{V}$ runs $\text{POTInitV}(1^\kappa, m_1, p_1, \dots, m_N, p_N, D_{max})$ in order to obtain a database commitment T and a key pair $(pk_\mathcal{V}, sk_\mathcal{V})$, and sends $(pk_\mathcal{V}, T)$ to $\mathcal{B}$. On input $(\text{deposit}, ac_0)$, $\mathcal{B}$ runs $(P, D'_0) \leftarrow \text{POTInitB}(1^\kappa, pk_\mathcal{V}, T, ac_0)$ and aborts if the output is **reject**. Otherwise, $\mathcal{B}$ sends the payment message (P) to $\mathcal{V}$ and pays an amount of ac_0 through an arbitrary payment channel. $\mathcal{V}$ runs $(D_0, ac_0) \leftarrow \text{POTGetDep}(sk_\mathcal{V}, P, D_{max})$ and checks that ac_0 corresponds to the amount of money received. $\mathcal{V}$ stores state information $V_0 = (T, sk_\mathcal{V}, pk_\mathcal{V}, D_0)$ and outputs $(\text{deposit}, ac_0)$, and $\mathcal{B}$ stores state information $B_0 = (pk_\mathcal{V}, T, D'_0)$.

Transfer phase. On input $(\text{request}, \tau_i)$, $\mathcal{B}$ runs $\text{POTReq}(pk_\mathcal{V}, T, D'_{i-1}, \tau_i)$ to get a request Q and private state (Q', D'_i) . $\mathcal{B}$ sends (Q) and stores (Q', D'_i) . On input $(\text{response}, b)$, if $b = 0$ $\mathcal{V}$ sends $(\perp)$ to $\mathcal{B}$. If $b = 1$, $\mathcal{V}$ runs $\text{POTVerReq}(pk_\mathcal{V}, D_{i-1}, Q)$ and ignores the request if the output is **reject**. Otherwise $\mathcal{V}$ runs $\text{POTResp}(pk_\mathcal{V}, sk_\mathcal{V}, Q)$ to obtain a response R and state D_i , and sends (R) to $\mathcal{B}$. $\mathcal{B}$ runs $\text{POTVerResp}(pk_\mathcal{V}, R)$ and outputs $(\text{response}, \perp)$ if the output is **reject**. Otherwise $\mathcal{B}$ runs $\text{POTComplete}(T, R, Q')$ to obtain m_{τ_i} . $\mathcal{V}$ stores state information $V_i = (sk_\mathcal{V}, T, pk_\mathcal{V}, D_i)$, and $\mathcal{B}$ stores state information $B_i = (T, pk_\mathcal{V}, D'_i)$ and outputs $(\text{response}, m_{\tau_i})$.

POTInitV($1^\kappa, m_1, p_1, \dots, m_N, p_N, D_{max}$) computes a pairing group setup $\Phi = (p, \mathbb{G}, \mathbb{G}_t, e, g)$, picks a random generator $h \in \mathbb{G}$ and sets $H = e(g, h)$. It runs InitVer($1^\kappa, D_{max}$) to obtain par_{ran} (which includes parameters of the commitment scheme par_{com}) and Kg(1^κ) to obtain (sk_{enc}, pk_{enc}) .² It also runs WKg(1^κ) to obtain a key pair (sk, pk) for the signature scheme, and, for $i = 1$ to N , it computes $A_i = \text{WSign}(sk, p_i)$ and $B_i = e(h, A_i) \cdot m_i$. It sets $C_i = (A_i, B_i, p_i)$. It computes a non-interactive proof $\pi_1 = \text{NIPK}\{(h) : H = e(g, h)\}$. It outputs $sk_{\mathcal{V}} = (sk_{enc}, h)$, $pk_{\mathcal{V}} = (\Phi, H, par_{ran}, pk, pk_{enc}, \pi_1)$ and $T = (C_1, \dots, C_N)$.

POTInitB($1^\kappa, pk_{\mathcal{V}}, T, ac_0$) parses, for $i = 1$ to N , C_i as (A_i, B_i, p_i) , and checks whether $\text{WVf}(pk, A_i, p_i)$ outputs **accept**. It also verifies π_1 and checks par_{ran} via InitP(par_{ran}). Then it runs Enc(pk_{enc}, ac_0) to obtain an encryption ct and computes a commitment $D_0 = \text{PECom}(par_{com}, ac_0, open_{ac_0})$ for random $open_{ac_0}$.³ It sets $P = (ct, open_{ac_0})$ and $D_0 = (ac_0, open_{ac_0}, D_0)$. It outputs (P, D_0) .

POTGetDep($sk_{\mathcal{V}}, P, D_{max}$) runs Dec(sk_{enc}, ct) to obtain ac_0 and checks that $ac_0 \in [0, D_{max}]$. It computes $D_0 = \text{PECom}(par_{com}, ac_0, open_{ac_0})$ and outputs (D_0, ac_0) .

POTReq($pk_{\mathcal{V}}, T, D'_{i-1}, \tau$) calculates $ac_i = ac_{i-1} - p_\tau$, picks random $(open_p, open_{ac_i}) \leftarrow \mathbb{Z}_p$, and computes $D_p = \text{PECom}(par_{com}, p_\tau, open_p)$ and $D_i = \text{PECom}(par_{com}, ac_i, open_{ac_i})$. It picks random $v \in \mathbb{Z}_p$ and computes $V = A_\tau^v$. It computes a non-interactive proof⁴ π_2 :

$$\text{NIPK}\{(p_\tau, open_p, ac_i, open_{ac_i}, v, \alpha) : \\ e(V, pk) = e(V, g)^{-p_\tau} e(g, g)^v \wedge \quad (1)$$

$$D_p = \text{PECom}(par_{com}, p_\tau, open_p) \wedge \quad (2)$$

$$D_i = \text{PECom}(par_{com}, ac_i, open_{ac_i}) \wedge \quad (3)$$

$$D_{i-1} / (D_i D_p) = \tilde{h}^\alpha \wedge \quad (4)$$

$$0 \leq ac_i < D_{max}\}. \quad (5)$$

Equation 1 proves that $V = A_\tau^v$ and that A_τ is a signature computed by $\mathcal{V}$. Equations 2 and 3 prove knowledge of the committed price p_τ and ac_i respectively, where p_τ is the value signed in A_τ . Equation 4 proves that $ac_i = ac_{i-1} - p_\tau$, and Equation 5 proves that ac_i is non-negative. The algorithm outputs $Q = (V, D_p, D_i, \pi_2)$, $Q' = (v, \tau)$ and $D'_i = (ac_i, open_{ac_i}, D_i)$.

POTVerReq($pk_{\mathcal{V}}, D_{i-1}, Q$) verifies π_2 and outputs either **accept** or **reject**.

POTResp($pk_{\mathcal{V}}, sk_{\mathcal{V}}, Q$) parses Q as (V, D_p, D_i, π_2) and computes $W = e(h, V)$ and a proof $\pi_3 = \text{NIPK}\{(h) : H = e(g, h) \wedge W = e(V, h)\}$ that W is computed correctly by using the request V and the secret key h . It outputs $R = (W, \pi_3)$ and D_i .

² We can employ any IND-CPA secure encryption scheme.

³ Although it would be possible to use a fixed value $open_{ac_0}$, this allows $\mathcal{B}$ to hide her deposit from the adjudicator in the OFPOT scheme.

⁴ $\tilde{h}$ is included in the parameters of the commitment scheme $par_{com} = (\tilde{g}, \tilde{h})$.

POTVerResp(pk_V, R) verifies π_3 and outputs either **accept** or **reject**.
 POTComplete(T, R, Q') parses Q' as (v, τ) , R as (W, π_3) and C_τ as (A_τ, B_τ, p_τ) .
 It outputs $m_\tau = B_\tau / (W^{1/v})$.

Theorem 1. *This POT scheme securely realizes $\mathcal{F}_{\text{POT}}$.*

We prove this theorem in Appendix A. As for efficiency, we note that the communication and computation cost of our protocol is roughly that of the OT scheme in [18], plus the overhead introduced by the range proof in [27]. We refer to [34] for a comparison of the efficiency of several OT schemes, and to [27] for efficiency measurements of the range proof.

3 An Optimistic Fair POT Scheme

3.1 Technical Preliminaries

Verifiably Encrypted Signatures. A verifiably encrypted signature (VES) scheme consists of algorithms (Kg, AdjKg, Sign, Vf, Create, VesVf, Adj). Algorithm Kg(1^κ) outputs a key pair (sk, pk) . AdjKg(1^κ) outputs a key pair (ask, apk) for the adjudicator. Sign(sk, m) computes a signature σ under sk on a message m . Vf(pk, σ, m) outputs **accept** if σ is a valid signature on m under pk . Create(sk, apk, m) outputs a verifiably encrypted signature ω on input the secret key sk , the adjudicator's public key apk and the message m . VesVf(pk, apk, ω, m) outputs **accept** if ω is a valid verifiably encrypted signature on input the public key pk , the adjudicator's public key apk and the message m . Adj(pk, ask, apk, ω, m) outputs an ordinary signature σ on m if ω is valid.

A VES scheme is complete if, for all key pairs (sk, pk) computed by Kg(1^κ) and for all key pairs (ask, apk) computed by AdjKg(1^κ), it holds that VesVf($pk, apk, Create(sk, apk, m), m$) outputs **accept** and algorithm Vf($pk, Adj(pk, ask, apk, Create(sk, apk, m), m)$) outputs **accept**.

Security for verifiably encrypted signatures [13, 15] is defined via four properties: unforgeability, opacity, extractability and abuse-freeness. Roughly speaking, unforgeability ensures that it is intractable to compute a verifiably encrypted signature on behalf of another user. Opacity ensures that nobody but the adjudicator and the signer can obtain an ordinary signature from a verifiably encrypted signature. Extractability means that the adjudicator is able to extract an ordinary signature from a valid verifiably encrypted signature with all but negligible probability. Abuse-freeness prevents an adversary that colludes with the adjudicator from forging verifiably encrypted signatures.

3.2 Definition

Previous work on defining optimistic fair exchange in the ideal-world/real-world paradigm only covers the case where buyer's privacy is not protected [36]. Therefore, we define an ideal functionality $\mathcal{F}_{\text{OFPOT}}$ for OFPOT that extends $\mathcal{F}_{\text{POT}}$. It describes the functionality provided by our construction when the adjudicator $\mathcal{A}$

is neutral, and we prove security under this restriction. (We explain more intuitively the functionality provided by $\mathcal{F}_{\text{OFFPOT}}$ in Section 3.3.) We also recall the description of a functionality $\mathcal{F}_{\text{CRS}}$ given in [37], which is used when the POT scheme used to instantiate the OFFPOT scheme operates in the $\mathcal{F}_{\text{CRS}}$ -hybrid model, where parties have access to an honestly-generated common reference string.

Functionality $\mathcal{F}_{\text{CRS}}$

Parameterized with a distribution D and a set of participants $\mathcal{P}$, on input (crs) from party P , if $P \notin \mathcal{P}$ it aborts. Otherwise, if there is no value r recorded, it picks $r \leftarrow D$ and records r . It sends (crs, r) to P .

Functionality $\mathcal{F}_{\text{OFFPOT}}$

Parameterized with $(N, l, p_{\max}, D_{\max})$, and running with a vendor $\mathcal{V}$, a buyer $\mathcal{B}$ and an adjudicator $\mathcal{A}$, $\mathcal{F}_{\text{OFFPOT}}$ works as follows:

- On input a message (init, $m_1, p_1, \dots, m_N, p_N$) from $\mathcal{V}$, it behaves as $\mathcal{F}_{\text{POT}}$.
- On input a message (deposit, ac_0) from $\mathcal{B}$, it behaves as $\mathcal{F}_{\text{POT}}$.
- On input a message (request, i, τ) from $\mathcal{B}$, where $\tau \in \{1, \dots, N\}$, if messages (init, $\dots$) or (deposit, $\dots$) were not received before, i does not equal the number of the transfer t , $ac_i - p_\tau < 0$, or $dispute_i = 1$, then it does nothing. Otherwise, it sends (request, i) to $\mathcal{V}$ and receives (response, i, b) in response. It stores (i, τ, b) . If $b = 0$, it sends (response, $i, \perp$) to $\mathcal{B}$. If $b = 1$, it sends (response, i, m_τ) to $\mathcal{B}$. $\mathcal{B}$ returns a message (complete, i, c), which is handed to $\mathcal{V}$. If $c = 1$, it sets $ac_i = ac_{i-1} - p_\tau$ and updates the number of transfers $t = t + 1$. If $c = 0$, it sets $dispute_i = 1$.
- On input a message (complain $_{\mathcal{V}}$, i) from $\mathcal{V}$, if $dispute_i = 0$ or $t \neq i$, then it sends (compInv $_{\mathcal{V}}$) to $\mathcal{V}$ and (complain $_{\mathcal{V}}$, i , inv) to $\mathcal{A}$. Otherwise it sets $ac_i = ac_{i-1} - p_\tau$, updates the number of transfers $t = t + 1$ and sends (compSolved $_{\mathcal{V}}$) to $\mathcal{V}$, (compResp $_{\mathcal{V}}$, i, m_τ) to $\mathcal{B}$ and (complain $_{\mathcal{V}}$, i , solved) to $\mathcal{A}$.
- On input a message (complain $_{\mathcal{B}}$, i, τ) from $\mathcal{B}$, if $t \neq i$ or if there exists a tuple (i, τ, b) and $\tau \neq \tau'$, then it sends (compInv $_{\mathcal{B}}$) to $\mathcal{B}$ and (complain $_{\mathcal{B}}$, i , inv) to $\mathcal{A}$. Otherwise it sends (complain $_{\mathcal{B}}$, i) to $\mathcal{V}$. Upon receiving (collaborate, i, b) from $\mathcal{V}$, if $b = 0$ it sends (guilty $_{\mathcal{V}}$) to $\mathcal{V}$ and $\mathcal{B}$ and (complain $_{\mathcal{B}}$, i , guilty $_{\mathcal{V}}$) to $\mathcal{A}$. If $b = 1$ it sets $ac_i = ac_{i-1} - p_\tau$, updates the number of transfers $t = t + 1$, sends (compResp $_{\mathcal{B}}$, i, m_τ) to $\mathcal{B}$, (compSolved $_{\mathcal{B}}$) to $\mathcal{V}$ and (complain $_{\mathcal{B}}$, i , solved) to $\mathcal{A}$.

3.3 Intuition Behind Our Construction

Our OFPOT scheme extends a full-simulation secure POT scheme with a fair exchange protocol based on verifiably encrypted signatures. In a nutshell, at a purchase phase, $\mathcal{B}$ computes her request following the POT scheme and a verifiably encrypted signature that signs the request. Then $\mathcal{V}$ computes a response following the POT scheme. If $\mathcal{B}$ is satisfied, $\mathcal{B}$ reveals a valid signature on her request, and otherwise $\mathcal{B}$ complains. $\mathcal{V}$ also complains when he computed his response honestly and $\mathcal{B}$ did not reveal a valid signature.

More concretely, the initialization phase follows that of the POT scheme. Additionally, $\mathcal{B}$ signs the payment message she sends to $\mathcal{V}$, $\mathcal{A}$ hands its public key to $\mathcal{V}$ and $\mathcal{B}$, and $\mathcal{A}$ receives the public keys of $\mathcal{V}$ and $\mathcal{B}$. At purchase phase i , $\mathcal{B}$ computes a request Q_i following algorithm `POTReq` of the POT scheme, and computes a verifiably encrypted signature ω_i on $H(i, Q_i)$, where H is a collision-resistant hash function. $\mathcal{V}$ rejects the request if i is not the valid purchase index, i.e., if $\mathcal{V}$ did not receive a signature σ_i on $H(i-1, Q_{i-1})$ or if $\mathcal{V}$ already received a signature on $H(i, Q'_i)$. Otherwise $\mathcal{V}$ computes a response R following algorithm `POTResp` of the POT scheme. Finally, $\mathcal{B}$ obtains the message m_τ by running algorithm `POTComplete` and reveals to $\mathcal{V}$ a signature σ_i on $H(i, Q_i)$.⁵

When $\mathcal{B}$ does not reveal σ_i , $\mathcal{V}$ complains by sending to $\mathcal{A}$ the request (i, Q_i, ω_i) and the response R . $\mathcal{A}$ verifies both request and response, sends R to $\mathcal{B}$, extracts σ_i from ω_i and sends σ_i to $\mathcal{V}$. Verification by $\mathcal{A}$ can be done without knowing the choice τ of $\mathcal{B}$. We note that a malicious $\mathcal{V}$ cannot produce fake requests on behalf of $\mathcal{B}$ thanks to the unforgeability property of the VES scheme. Moreover, if $\mathcal{B}$ did not reveal σ_i because the response R' from $\mathcal{V}$ was not correct, after $\mathcal{V}$ complains $\mathcal{A}$ ensures that $\mathcal{B}$ receives a valid response.

When the response sent by $\mathcal{V}$ is not correct, $\mathcal{B}$ complains by sending to $\mathcal{A}$ a request (i, Q_i, ω_i) .⁶ $\mathcal{A}$ tells $\mathcal{V}$ to send (j, Q_j, σ_j) for $j = i - 1$ in order to check if the complaint is valid. (When $i = 1$, $\mathcal{V}$ should send the signature on the payment message.) If $\mathcal{V}$ sends it for $j \geq i$, then $\mathcal{A}$ states that the complaint is invalid because this means that $\mathcal{B}$ has already shown conformity for purchase i , and if $j < i - 1$, then it asks $\mathcal{B}$ to send a request with index j .⁷ If $\mathcal{V}$ does not reveal any signature, $\mathcal{A}$ finds $\mathcal{V}$ guilty. (We note that $\mathcal{V}$ cannot produce signatures on behalf of $\mathcal{B}$ thanks to the opacity property of the VES scheme.) Otherwise, $\mathcal{A}$ verifies the request from $\mathcal{B}$ and sends it to $\mathcal{V}$. If $\mathcal{V}$ returns a different tuple (i, Q'_i, ω'_i) , then $\mathcal{A}$ states that the complaint is invalid because this may mean that a malicious $\mathcal{B}$ already obtained the message requested in Q'_i , and is trying to obtain a different message. Otherwise, if $\mathcal{V}$ returns a valid response R , $\mathcal{A}$ forwards R to $\mathcal{B}$ and reveals σ_i to $\mathcal{V}$. If not, $\mathcal{A}$ finds $\mathcal{V}$ guilty.

⁵ In practical implementations, this message can be sent together with the next request in order to save communication rounds.

⁶ We note that $\mathcal{B}$ can complain without previously interacting with $\mathcal{V}$, but this does not give a malicious $\mathcal{B}$ any advantage over $\mathcal{V}$.

⁷ We note that $\mathcal{V}$ should always send the larger index j for which he possesses a signature, because otherwise he would be claiming that some purchases did not happen, thus increasing the account of $\mathcal{B}$.

We prove that our scheme realizes $\mathcal{F}_{\text{OFFPOT}}$ if it is instantiated with a full-simulation secure POT scheme (which realizes $\mathcal{F}_{\text{POT}}$) and under the assumptions that the VES scheme is unforgeable, opaque and extractable, and that $\mathcal{A}$ is neutral. The abuse-freeness property of the VES scheme gives $\mathcal{B}$ some protection when $\mathcal{V}$ and $\mathcal{A}$ collude: they cannot produce fake requests. However, once $\mathcal{B}$ sends a request, a malicious $\mathcal{A}$ can reveal σ_i to $\mathcal{V}$ without sending R to $\mathcal{B}$. (Nevertheless, privacy of $\mathcal{B}$ is still preserved.) Similarly, when $\mathcal{B}$ and $\mathcal{A}$ collude, $\mathcal{V}$ is not protected.

When our construction is instantiated with the universally composable POT scheme in [7] (which we summarize in Appendix C), we obtain a UC secure construction in the common reference string model. (In the description given below, we include the common reference string as input of the algorithms, but it should be removed when unnecessary.) When instantiated with the POT scheme proposed in Section 2, we obtain a full-simulation secure and more efficient scheme.

Finally, we point out that our scheme does not ensure that all the buyers receive the same messages, or that the messages that $\mathcal{V}$ sells fulfill the expectations of $\mathcal{B}$. Nevertheless, there already exist countermeasures which employ a third party against these problems [38], and $\mathcal{A}$ can apply them. Basically, these countermeasures ensure that all the buyers obtain the same messages from $\mathcal{V}$. We can achieve this by ensuring that they obtain the same database commitment T .

3.4 Description of the Scheme

We begin with a high level description of the OFFPOT scheme. Details on the algorithms can be found below. We recall that the scheme is parameterized with $(N, l, p_{\max}, D_{\max})$.

OFFPOT

Initialization phase. On input $(\text{init}, m_1, p_1, \dots, m_N, p_N)$, $\mathcal{V}$ queries $\mathcal{F}_{\text{CRS}}$ with (crs) , which runs $\text{POTGenCRS}(1^\kappa, p_{\max}, D_{\max})$ and returns (crs, crs) . $\mathcal{B}$ and $\mathcal{A}$ also query $\mathcal{F}_{\text{CRS}}$ with (crs) , which returns (crs, crs) . $\mathcal{V}$ runs $\text{POTInitV}(\text{crs}, m_1, p_1, \dots, m_N, p_N, D_{\max})$ to get a database commitment T and a key pair $(sk_{\mathcal{V}}, pk_{\mathcal{V}})$, and sends $(pk_{\mathcal{V}}, T)$ to $\mathcal{B}$. On input $(\text{deposit}, ac_0)$, $\mathcal{B}$ computes $(P, D'_0) \leftarrow \text{POTInitB}(\text{crs}, pk_{\mathcal{V}}, T, ac_0)$ to obtain a payment message P and the opening of the commitment to the account D'_0 , and aborts if the output is **reject**. Otherwise, $\mathcal{B}$ runs $(sk_{\mathcal{B}}, pk_{\mathcal{B}}, \sigma_0) \leftarrow \text{OFlnitB}(\text{crs}, D'_0)$ to obtain a key pair $(sk_{\mathcal{B}}, pk_{\mathcal{B}})$ and a signature σ_0 on D_0 , sends $(P, pk_{\mathcal{B}}, \sigma_0)$ and pays ac_0 to $\mathcal{V}$ through an arbitrary payment channel. $\mathcal{V}$ runs $(D_0, ac_0) \leftarrow \text{POTGetDep}(\text{crs}, sk_{\mathcal{V}}, P, D_{\max})$ to obtain the commitment D_0 and checks that ac_0 equals the amount of money paid. $\mathcal{V}$ runs $\text{OFlnitV}(pk_{\mathcal{B}}, \sigma_0, D_0)$

to check σ_0 and aborts if the output is reject. $\mathcal{A}$ runs $(ask, apk) \leftarrow \text{OFInitA}(crs)$ and sends apk to $\mathcal{V}$ and $\mathcal{B}$. $\mathcal{V}$ and $\mathcal{B}$ retrieve apk . $\mathcal{V}$ ($\mathcal{B}$) sends $(pk_{\mathcal{V}}, pk_{\mathcal{B}})$ ($(pk'_{\mathcal{V}}, pk'_{\mathcal{B}})$) to $\mathcal{A}$, $\mathcal{A}$ checks equality of the public keys and stores $(pk_{\mathcal{V}}, pk_{\mathcal{B}}, ask)$. (Alternatively, $\mathcal{A}$, $\mathcal{V}$ and $\mathcal{B}$ can use a PKI.) $\mathcal{V}$ stores state information $V_0 = (sk_{\mathcal{V}}, T, pk_{\mathcal{V}}, pk_{\mathcal{B}}, apk, D_0, \sigma_0)$ and outputs $(\text{deposit}, ac_0)$, and $\mathcal{B}$ stores state information $B_0 = (sk_{\mathcal{B}}, T, pk_{\mathcal{B}}, pk_{\mathcal{V}}, apk, D'_0)$ and outputs $(\text{init}, p_1, \dots, p_N)$.

Transfer phase. On input $(\text{request}, i, \tau)$, $\mathcal{B}$ does nothing if $\mathcal{B}$ has previously received $(\text{request}, i, \dots)$. Otherwise $\mathcal{B}$ runs $(Q_i, Q'_i, D'_i) \leftarrow \text{POTReq}(crs, pk_{\mathcal{V}}, T, D'_{i-1}, \tau)$ to compute a request message Q_i , trapdoor information Q'_i and opening D'_i , and $\omega_i \leftarrow \text{OFReq}(sk_{\mathcal{B}}, apk, i, Q_i)$ to obtain a verifiably encrypted signature ω_i on Q_i . $\mathcal{B}$ sends the request (i, Q_i, ω_i) and stores (Q'_i, D'_i) . Upon receiving (i, Q_i, ω_i) , if i does not equal the number of the transfer t , $\mathcal{V}$ does nothing. Otherwise, on input $(\text{response}, b)$, if $b = 0$ $\mathcal{V}$ sends $(\perp)$ to $\mathcal{B}$. If $b = 1$ $\mathcal{V}$ runs $\text{POTVerReq}(crs, pk_{\mathcal{V}}, D_{i-1}, Q_i)$ and $\text{OFVerReq}(pk_{\mathcal{B}}, apk, \omega_i, i, Q_i)$. If any of the two algorithms outputs reject, $\mathcal{V}$ rejects the request. Otherwise $\mathcal{V}$ runs $(R) \leftarrow \text{POTResp}(crs, pk_{\mathcal{V}}, sk_{\mathcal{V}}, Q_i)$, sends the response (R) and keeps state (i, Q_i, ω_i) . If the output of $\text{POTVerReq}(crs, pk_{\mathcal{V}}, R)$ is reject, $\mathcal{B}$ outputs $(\text{response}, i, \perp)$ and proceeds with the complaint phase. Otherwise, $\mathcal{B}$ runs $(m_{\tau}) \leftarrow \text{POTComplete}(crs, T, R, Q'_i)$ and outputs $(\text{response}, i, m_{\tau})$. On input $(\text{complete}, c, i)$, if $c = 0$, $\mathcal{B}$ sends $(\perp)$, and, if $c = 1$, $\mathcal{B}$ runs $\sigma_i \leftarrow \text{OFComplete}(sk_{\mathcal{B}}, i, Q_i)$ and sends (σ_i) . $\mathcal{B}$ stores state information $B_i = (sk_{\mathcal{B}}, T, pk_{\mathcal{B}}, pk_{\mathcal{V}}, apk, D'_i)$. $\mathcal{V}$ runs $\text{OFVerComp}(pk_{\mathcal{B}}, \sigma_i, i, Q_i)$ and, if the output is reject, $\mathcal{V}$ outputs $(\text{complete}, i, 0)$ and proceeds with the complaint phase. Otherwise $\mathcal{V}$ stores state information $V_i = (sk_{\mathcal{V}}, T, pk_{\mathcal{V}}, pk_{\mathcal{B}}, apk, Q_i, \sigma_i)$, increments the number of the transfer $t = t + 1$ and outputs $(\text{complete}, i, 1)$.

Vendor complaint. On input the tuple $(\text{complain}_{\mathcal{V}}, i)$, if no valid request (i, Q_i, ω_i) was previously received, $\mathcal{V}$ does nothing. Otherwise $\mathcal{V}$ runs $\text{POTResp}(crs, pk_{\mathcal{V}}, sk_{\mathcal{V}}, Q_i)$ and sends the request-response pair $((i, Q_i, \omega_i), (R))$ to $\mathcal{A}$, along with the signature of the previous transfer $(i - 1, Q_{i-1}, \sigma_{i-1})$ ($(0, D_0, \sigma_0)$ when $i = 1$). $\mathcal{A}$ parses Q_{i-1} to obtain D_{i-1} and runs $\text{POTVerReq}(crs, pk_{\mathcal{V}}, D_{i-1}, Q_i)$, $\text{OFVerReq}(pk_{\mathcal{B}}, apk, \omega_i, i, Q_i)$, $\text{OFVerComp}(pk_{\mathcal{B}}, \sigma_{i-1}, i - 1, Q_{i-1})$ ($\text{OFInitV}(pk_{\mathcal{B}}, \sigma_0, D_0)$ when $i = 1$) and $\text{POTVerResp}(crs, pk_{\mathcal{V}}, R)$ and, if any of them outputs reject, sets $\sigma_i = \perp$ and $R = \perp$ and outputs $(\text{complain}_{\mathcal{V}}, i, \text{inv})$. Otherwise $\mathcal{A}$ runs $\sigma_i \leftarrow \text{OFAdj}(pk_{\mathcal{B}}, ask, apk, \omega_i, i, Q_i)$. $\mathcal{A}$ sends σ_i to $\mathcal{V}$ and (i, R) to $\mathcal{B}$ and outputs $(\text{complain}_{\mathcal{V}}, i, \text{solved})$. $\mathcal{V}$ runs $\text{OFVerComp}(pk_{\mathcal{B}}, \sigma_i, i, Q_i)$ and outputs $(\text{compSolved}_{\mathcal{V}})$ if the output is accept. $\mathcal{B}$ runs $(m_{\tau}) \leftarrow \text{POTComplete}(crs, T, R, Q'_i)$, where Q'_i corresponds to the request for transfer i , and outputs $(\text{compResp}_{\mathcal{V}}, i, m_{\tau})$.

Buyer complaint. On input $(\text{complain}_{\mathcal{B}}, i, \tau)$, if a request (i, Q_i, ω_i) was previously computed $\mathcal{B}$ sends it to $\mathcal{A}$. Otherwise, $\mathcal{B}$ sends to $\mathcal{A}$ a new re-

quest computed by running POTReq and OFReq. $\mathcal{A}$ sends (i) to $\mathcal{V}$, which returns (j, Q_j, σ_j) ($((0, D_0, \sigma_0)$ if $i = 1$). Then $\mathcal{A}$ proceeds as follows:

- If σ_j is invalid, $\mathcal{A}$ sends $(\text{guilty}_{\mathcal{V}})$ to $\mathcal{V}$ and $\mathcal{B}$ and outputs $(\text{complain}_{\mathcal{B}}, i, \text{guilty}_{\mathcal{V}})$.
- If σ_j is valid but $j \geq i$, $\mathcal{A}$ sends $(\text{compInv}_{\mathcal{B}})$ to $\mathcal{B}$ and outputs $(\text{complain}_{\mathcal{B}}, i, \text{inv})$.
- If σ_j is valid and $j = i - 1$, $\mathcal{A}$ parses Q_{i-1} to obtain D_{i-1} , runs $\text{POTVerReq}(crs, pk_{\mathcal{V}}, D_{i-1}, Q_i)$ and, if the output is `reject`, sends $(\text{compInv}_{\mathcal{B}})$ to $\mathcal{B}$ and outputs $(\text{complain}_{\mathcal{B}}, i, \text{inv})$. $\mathcal{V}$ (possibly) sends another request (i, Q'_i, ω'_i) . $\mathcal{A}$ verifies it and, if it is correct, but $(i, Q'_i, \omega'_i) \neq (i, Q_i, \omega_i)$, $\mathcal{A}$ sends $(\text{compInv}_{\mathcal{B}})$ to $\mathcal{B}$ and outputs $(\text{complain}_{\mathcal{B}}, i, \text{inv})$. Otherwise $\mathcal{A}$ sends (i, Q_i, ω_i) to $\mathcal{V}$. $\mathcal{V}$ verifies the request, runs $(R) \leftarrow \text{POTResp}(crs, T, sk_{\mathcal{V}}, Q_i)$ and sends (R) to $\mathcal{A}$. $\mathcal{A}$ verifies the response via $\text{POTVerResp}(crs, pk_{\mathcal{V}}, R)$ and, if it is not valid, $\mathcal{A}$ sends $(\text{guilty}_{\mathcal{V}})$ to $\mathcal{B}$ and $\mathcal{V}$ and outputs $(\text{complain}_{\mathcal{B}}, i, \text{guilty}_{\mathcal{V}})$. If it is valid, $\mathcal{A}$ sends R to $\mathcal{B}$ and $\sigma_i \leftarrow \text{OFAdj}(pk_{\mathcal{B}}, ask, apk, \omega_i, i, Q_i)$ to $\mathcal{V}$, and outputs $(\text{complain}_{\mathcal{B}}, i, \text{solved})$. $\mathcal{B}$ retrieves m_{τ} via POTComplete and outputs $(\text{compResp}_{\mathcal{B}}, i, m_{\tau})$, while $\mathcal{V}$ outputs $\text{compSolved}_{\mathcal{B}}$.

Algorithms POT* correspond to those of the POT schemes (see Section 2.4 and Appendix C). Algorithms OF* are defined as follows:

OFInitB(crs, D'_0) chooses a collision resistant hash function $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p$ and runs $(sk, pk) \leftarrow \text{Kg}(1^\kappa)$ of the VES scheme for the pairing group setup contained in crs . Then it parses D'_0 to obtain the commitment to the deposit D_0 and signs $\sigma_0 \leftarrow \text{Sign}(sk, H(0, D_0))$. It outputs $sk_{\mathcal{B}} = sk$, $pk_{\mathcal{B}} = (H, pk)$ and σ_0 .

OFInitV($pk_{\mathcal{B}}, \sigma_0, D_0$) outputs the result of $\text{Vf}(pk, \sigma_0, H(0, D_0))$.

OFInitA(crs) runs $\text{AdjKg}(1^\kappa)$ and outputs a key pair (ask, apk) for the pairing group setup contained in crs .

OFReq($sk_{\mathcal{B}}, apk, i, Q_i$) outputs a verifiably encrypted signature $\omega_i \leftarrow \text{Create}(sk_{\mathcal{B}}, apk, H(i, Q_i))$.

OFVerReq($pk_{\mathcal{B}}, apk, \omega_i, i, Q_i$) outputs the result of $\text{VesVf}(pk, apk, \omega_i, H(i, Q_i))$.

OFComplete($sk_{\mathcal{B}}, i, Q_i$) outputs $\sigma_i \leftarrow \text{Sign}(sk_{\mathcal{B}}, H(i, Q_i))$.

OFVerComp($pk_{\mathcal{B}}, \sigma_i, i, Q_i$) outputs $\text{Vf}(pk, \sigma_i, H(i, Q_i))$.

OFAdj($pk_{\mathcal{B}}, ask, apk, \omega_i, i, Q_i$) outputs $\text{Adj}(pk, ask, apk, \omega, H(i, Q_i))$.

Theorem 2. *This OFPOT scheme securely realizes $\mathcal{F}_{\text{OFPOT}}$.*

We prove this theorem in Appendix B. As for efficiency, we note that the overhead in terms of computation and communication cost introduced by the VES scheme are small compared to the cost of the POT scheme. Therefore, a secure POT can efficiently be extended to provide optimistic fair exchange.

4 Conclusion

Our contribution is twofold. First, we have designed a full-simulation secure POT scheme that is more efficient than previous work. Second, we have proposed a generic construction that provides any secure POT scheme with optimistic fair exchange.

We leave open the definition of a more general ideal functionality for fair privacy-preserving e-commerce protocols. Another interesting open problem is the design of a fair POT scheme that is abuse-free in the sense of [39]. Further research also needs to be conducted to show how to integrate e-commerce protocols based on POT with digital rights management systems, and to analyze the compliance of such e-commerce protocols with e-commerce legislation.

Acknowledgements

This work was supported in part by the IAP Programme P6/26 BCRYPT of the Belgian State. Alfredo Rial is funded by the Research Foundation - Flanders (FWO).

References

1. Ackerman, M.S., Cranor, L.F., Reagle, J.: Privacy in e-commerce: examining user scenarios and privacy preferences. In: ACM Conference on Electronic Commerce. (1999) 1–8
2. Tsai, J., Egelman, S., Cranor, L., Acquisti, R.: The effect of online privacy information on purchasing behavior: An experimental study, working paper. (June 2007)
3. : Enforcing privacy promises: Section 5 of the ftc act. Federal Trade Commission Act <http://www.ftc.gov/privacy/privacyinitiatives/promises.html>.
4. Kremer, S.: Formal analysis of optimistic fair exchange protocols (2004)
5. Aiello, W., Ishai, Y., Reingold, O.: Priced oblivious transfer: How to sell digital goods. In Pfitzmann, B., ed.: EUROCRYPT. Volume 2045 of Lecture Notes in Computer Science., Springer (2001) 119–135
6. Tobias, C.: Practical oblivious transfer protocols. In Petitcolas, F.A.P., ed.: Information Hiding. Volume 2578 of Lecture Notes in Computer Science., Springer (2002) 415–426
7. Rial, A., Kohlweiss, M., Preneel, B.: Universally composable adaptive priced oblivious transfer. [40] 231–247
8. Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. Commun. ACM **28**(6) (1985) 637–647
9. Goldreich, O.: A simple protocol for signing contracts. In: CRYPTO. (1983) 133–136
10. Bao, F., Deng, R.H., Mao, W.: Efficient and practical fair exchange protocols with off-line ttp. In: IEEE Symposium on Security and Privacy, IEEE Computer Society (1998) 77–85
11. Asokan, N., Shoup, V., Waidner, M.: Optimistic fair exchange of digital signatures (extended abstract). In: EUROCRYPT. (1998) 591–606

12. Avoine, G., Vaudenay, S.: Optimistic fair exchange based on publicly verifiable secret sharing. In Wang, H., Pieprzyk, J., Varadharajan, V., eds.: ACISP. Volume 3108 of Lecture Notes in Computer Science., Springer (2004) 74–85
13. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In Biham, E., ed.: EUROCRYPT. Volume 2656 of Lecture Notes in Computer Science., Springer (2003) 416–432
14. Lu, S., Ostrovsky, R., Sahai, A., Shacham, H., Waters, B.: Sequential aggregate signatures and multisignatures without random oracles. In Vaudenay, S., ed.: EUROCRYPT. Volume 4004 of Lecture Notes in Computer Science., Springer (2006) 465–485
15. Rückert, M., Schröder, D.: Security of verifiably encrypted signatures and a construction without random oracles. [40] 17–34
16. Ray, I., Ray, I.: An anonymous fair exchange e-commerce protocol. In: IPDPS, IEEE Computer Society (2001) 172
17. Naor, M., Pinkas, B.: Computationally secure oblivious transfer. *J. Cryptology* **18**(1) (2005) 1–35
18. Camenisch, J., Neven, G., Shelat, A.: Simulatable adaptive oblivious transfer. In Naor, M., ed.: EUROCRYPT. Volume 4515 of Lecture Notes in Computer Science., Springer (2007) 573–590
19. Boneh, D., Boyen, X.: Short signatures without random oracles. [41] 56–73
20. Camenisch, J., Dubovitskaya, M., Neven, G.: Oblivious transfer with access control. In Al-Shaer, E., Jha, S., Keromytis, A.D., eds.: ACM Conference on Computer and Communications Security, ACM (2009) 131–140
21. Boneh, D., Boyen, X., Goh, E.J.: Hierarchical identity based encryption with constant size ciphertext. In Cramer, R., ed.: EUROCRYPT. Volume 3494 of Lecture Notes in Computer Science., Springer (2005) 440–456
22. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.* **17**(2) (1988) 281–308
23. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: CRYPTO '92. Volume 576 of LNCS. (1992) 129–140
24. Bellare, M., Goldreich, O.: On defining proofs of knowledge. In Brickell, E.F., ed.: CRYPTO '92. Volume 740 of LNCS., Springer-Verlag (1992) 390–420
25. Schnorr, C.P.: Efficient signature generation for smart cards. *Journal of Cryptology* **4**(3) (1991) 239–252
26. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: CRYPTO '92. Volume 740 of LNCS. (1993) 89–105
27. Camenisch, J., Chaabouni, R., Shelat, A.: Efficient protocols for set membership and range proofs. [51] 234–252
28. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: CRYPTO '94. Volume 839 of LNCS. (1994) 174–187
29. Damgård, I.: Concurrent zero-knowledge is easy in practice. Available online at Theory of Cryptography Library (June 1999)
30. Damgård, I.: On σ -protocols. Available at <http://www.daimi.au.dk/~ivan/Sigma.ps> (2002)
31. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In Odlyzko, A.M., ed.: CRYPTO '86. Volume 263 of LNCS., Springer Verlag (1987) 186–194
32. Camenisch, J., Stadler, M.: Proof systems for general statements about discrete logarithms. Technical Report TR 260, Institute for Theoretical Computer Science, ETH Zürich (March 1997)

33. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS). (2001) 136–145
34. Green, M., Hohenberger, S.: Universally composable adaptive oblivious transfer. [51] 179–197
35. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: First ACM Conference on Computer and Communication Security, Association for Computing Machinery (1993) 62–73
36. Okada, Y., Manabe, Y., Okamoto, T.: An optimistic fair exchange protocol and its security in the universal composability framework. *IJACT* **1**(1) (2008) 70–77
37. Canetti, R.: Obtaining universally composable security: Towards the bare bones of trust. In: ASIACRYPT. (2007) 88–112
38. Herranz, J.: Restricted adaptive oblivious transfer. Cryptology ePrint Archive, Report 2008/182 (2008) <http://eprint.iacr.org/>.
39. Garay, J., Jakobsson, M., MacKenzie, P.: Abuse-free optimistic contract signing. In Wiener, M., ed.: CRYPTO '99. Volume 1666 of LNCS., Springer Verlag (1999) 499–466
40. Shacham, H., Waters, B., eds.: Pairing-Based Cryptography - Pairing 2009, Third International Conference, Palo Alto, CA, USA, August 12-14, 2009, Proceedings. In Shacham, H., Waters, B., eds.: Pairing. Volume 5671 of Lecture Notes in Computer Science., Springer (2009)
41. Cachin, C., Camenisch, J., eds.: Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings. In Cachin, C., Camenisch, J., eds.: EUROCRYPT. Volume 3027 of Lecture Notes in Computer Science., Springer (2004)
42. Boyen, X., Waters, B.: Full-domain subgroup hiding and constant-size group signatures. In Okamoto, T., Wang, X., eds.: Public Key Cryptography. Volume 4450 of Lecture Notes in Computer Science., Springer (2007) 1–15
43. Belenkiy, M., Chase, M., Kohlweiss, M., Lysyanskaya, A.: P-signatures and noninteractive anonymous credentials. In Canetti, R., ed.: TCC. Volume 4948 of Lecture Notes in Computer Science., Springer (2008) 356–374
44. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In Franklin, M.K., ed.: CRYPTO. Volume 3152 of Lecture Notes in Computer Science., Springer (2004) 41–55
45. Santis, A.D., Di Crescenzo, G., Persiano, G.: Necessary and sufficient assumptions for non-interactive zero-knowledge proofs of knowledge for all NP relations. In Montanari, U., Rolim, J.P., Welzl, E., eds.: Proc. 27th International Colloquium on Automata, Languages and Programming (ICALP). Volume 1853 of LNCS., Springer Verlag (2000) 451–462
46. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. *SIAM J. Comput.* **18**(1) (1989) 186–208
47. Goldreich, O.: Foundations of Cryptography: Basic Tools. Cambridge University Press, New York, NY, USA (2000)
48. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications. In: STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing, New York, NY, USA, ACM Press (1988) 103–112
49. Feige, U., Lapidot, D., Shamir, A.: Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM Journal on Computing* **29**(1) (1999) 1–28

50. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In Smart, N.P., ed.: EUROCRYPT. Volume 4965 of Lecture Notes in Computer Science., Springer (2008) 415–432
51. Pieprzyk, J., ed.: Advances in Cryptology - ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, December 7-11, 2008. Proceedings. In Pieprzyk, J., ed.: ASIACRYPT. Volume 5350 of Lecture Notes in Computer Science., Springer (2008)

A Security Proof of POT scheme

In order to prove Theorem 1, we need to build a simulator $\mathcal{S}' = \{\mathcal{S}'_{\mathcal{V}}, \mathcal{S}'_{\mathcal{B}}\}$ that invokes a copy of adversary $\mathcal{E}$ and interacts with $\mathcal{F}_{\text{POT}}$ and environment $\mathcal{Z}$ in such a way that ensembles $\text{IDEAL}_{\mathcal{F}_{\text{POT}}, \mathcal{S}', \mathcal{Z}}$ and $\text{REAL}_{\text{POT}, \mathcal{E}, \mathcal{Z}}$ are computationally indistinguishable.

Simulation of buyer's security. In this case only the vendor $\mathcal{V}$ is corrupted.

1. Upon receiving message $(pk_{\mathcal{V}}, T)$ from $\mathcal{E}$, $\mathcal{S}'_{\mathcal{V}}$ verifies $pk_{\mathcal{V}}$ and T as described in **POTInitB**. Then $\mathcal{S}'_{\mathcal{V}}$ parses $pk_{\mathcal{V}}$ as $(\Phi, H, par_{\text{ran}}, pk, pk_{\text{enc}}, \pi_1)$ and extracts the witness h from π_1 (via forking lemma). $\mathcal{S}'_{\mathcal{V}}$ parses T as $(C_1, \dots, C_N)$ and, for $i = 1$ to N , parses C_i as (A_i, B_i, p_i) and computes $m_i = B_i/e(h, A_i)$. $\mathcal{S}'_{\mathcal{V}}$ sends $(\text{init}, m_1, p_1, \dots, m_N, p_N)$ to $\mathcal{F}_{\text{POT}}$.
2. Upon receiving $(\text{deposit}, ac_0)$ from $\mathcal{F}_{\text{POT}}$, $\mathcal{S}'_{\mathcal{V}}$ picks $open_{ac_0}$, sets $P = (\text{Enc}(pk_{\text{enc}}, ac_0), open_{ac_0})$, stores $D'_0 = (ac_0, open_{ac_0}, D_0 = \text{PECom}(par_{\text{com}}, ac_0, open_{ac_0}))$ and sends (P) to $\mathcal{E}$.
3. Upon receiving (request) from $\mathcal{F}_{\text{POT}}$, $\mathcal{S}'_{\mathcal{V}}$ selects $\tau_{\min}$ that corresponds to the item with the lowest price, and runs **POTReq** $(pk_{\mathcal{V}}, T, D'_{i-1}, \tau_{\min})$ to obtain Q, Q' and D'_i . $\mathcal{S}'_{\mathcal{V}}$ sends (Q) to $\mathcal{E}$.
4. Upon receiving (R) from $\mathcal{E}$, $\mathcal{S}'_{\mathcal{V}}$ verifies π_3 . If the proof is correct, $\mathcal{S}'_{\mathcal{V}}$ sends $(\text{response}, 1)$ to $\mathcal{F}_{\text{POT}}$. Otherwise $\mathcal{S}'_{\mathcal{V}}$ sends $(\text{response}, 0)$ to $\mathcal{F}_{\text{POT}}$.

Claim. When only $\mathcal{V}$ is corrupted, the distribution ensembles $\text{IDEAL}_{\mathcal{F}_{\text{POT}}, \mathcal{S}'_{\mathcal{V}}, \mathcal{Z}}$ and $\text{REAL}_{\text{POT}, \mathcal{E}, \mathcal{Z}}$ are indistinguishable under the hiding property of the commitment scheme.

Proof. We show by means of a series of hybrid games that the environment $\mathcal{Z}$ cannot distinguish between the real execution ensemble $\text{REAL}_{\text{POT}, \mathcal{E}, \mathcal{Z}}$ and the simulated ensemble $\text{IDEAL}_{\mathcal{F}_{\text{POT}}, \mathcal{S}'_{\mathcal{V}}, \mathcal{Z}}$ with non-negligible probability. We denote by $\Pr [\mathbf{Game} \ i]$ the probability that $\mathcal{Z}$ distinguishes between the ensemble of **Game** i and that of the real execution.

Game 0: This game corresponds to the execution of the real-world protocol with an honest $\mathcal{B}$. Therefore, $\Pr [\mathbf{Game} \ 0] = 0$.

Game 1: This game proceeds as **Game 0**, except that we use the knowledge extractor to obtain h from π_1 . If extraction fails, we abort. The probability that extraction fails is negligible, and thus $|\Pr [\mathbf{Game} \ 1] - \Pr [\mathbf{Game} \ 0]| = \nu_1$.

Game 2: This game proceeds as **Game 1**, except that $P = \text{Enc}(pk_{enc}, ac_0)$ is replaced by another valid encryption of the same value under the same public key. Therefore $|\Pr [\text{Game 2}] - \Pr [\text{Game 1}]| = 0$.

Game 3: This game proceeds as **Game 2**, except that buyer's requests are replaced by other valid requests that use a fixed τ_{min} , which corresponds to the lowest price. Therefore $Q = (V, D_p, D_i, \pi_2)$ is computed as follows: $V = A_{\tau_{min}}^v$ for a random v ; $D_p = \text{PECom}(par_{com}, p_\tau, open_p)$; and $D_i = \text{PECom}(par_{com}, ac_i, open_{ac_i})$ for $ac_i = ac_{i-1} - p_\tau$. We note that V completely hides $A_{\tau_{min}}$ and that D_p and D_i completely hide p and ac_i because Pedersen commitments are information theoretically hiding. Moreover, the proof of knowledge π_2 is zero-knowledge and therefore does not leak any information about the witness. Therefore, $|\Pr [\text{Game 3}] - \Pr [\text{Game 2}]| = 0$.

$\mathcal{S}'_V$ performs all the changes described in **Game 3**, and forwards and receives messages from $\mathcal{F}_{POT}$ as described in our simulation. The distribution produced in **Game 3** is identical to that of our simulation. Therefore, by summation we have that $|\Pr [\text{Game 3}]| \leq \nu_1$.

Simulation of vendor's security. In this case only the buyer $\mathcal{B}$ is corrupted.

1. Upon receiving $(\text{init}, p_1, \dots, p_N)$ from $\mathcal{F}_{POT}$, for $i = 1$ to N , $\mathcal{S}'_B$ chooses random messages $m'_i \leftarrow \mathbb{G}_t$ and follows algorithm $\text{POTInitV}(1^\kappa, m'_1, p_1, \dots, m'_N, p_N, D_{max})$ to compute a database commitment T , pk_V and sk_V . $\mathcal{S}'_B$ sends (pk_V, T) to $\mathcal{E}$.
2. Upon receiving (P) from $\mathcal{E}$, $\mathcal{S}'_B$ follows algorithm $\text{POTGetDep}(sk_V, P, D_{max})$ to obtain (D_0, ac_0) and sends $(\text{deposit}, ac_0)$ to $\mathcal{F}_{POT}$.
3. Upon receiving (Q) , $\mathcal{S}'_B$ parses Q as (V, D_p, D_i, π_2) . $\mathcal{S}'_B$ verifies π_2 and extracts the witness $(p_\tau, open_p, ac_i, open_{ac_i}, ac_{i-1}, open_{ac_{i-1}}, v, \alpha)$. If p_τ is not a valid price, $\mathcal{S}'_B$ aborts. Otherwise $\mathcal{S}'_B$ maps p_τ to τ . Then $\mathcal{S}'_B$ checks that the extracted ac_i is correct, i.e., that it equals $ac_{i-1} - p_\tau$, and aborts if it is not the case. $\mathcal{S}'_B$ sends $(\text{request}, \tau)$ to $\mathcal{F}_{POT}$.
4. Upon receiving $(\text{response}, m_\tau)$ from $\mathcal{F}_{POT}$, $\mathcal{S}'_B$ parses T as $(C_1, \dots, C_N)$ and C_τ as (A_τ, B_τ, p_τ) . $\mathcal{S}'_B$ sets $W = (B_\sigma / m_\tau)^v$ and computes a simulated proof π_3 . $\mathcal{S}'_B$ sends $R = (W, \pi_3)$ to $\mathcal{E}$.

Theorem 3. *When only the buyer $\mathcal{B}$ is corrupted, the distribution ensembles $\text{IDEAL}_{\mathcal{F}_{POT}, \mathcal{S}'_B, \mathcal{Z}}$ and $\text{REAL}_{\text{POT}, \mathcal{E}, \mathcal{Z}}$ are computationally indistinguishable under the $(N+1, d+1)$ -SDH assumption and the $(N+1)$ -PDDH assumption, and under the binding property of the commitment scheme.*

Proof.

Game 0: This game corresponds to the execution of the real-world protocol with an honest $\mathcal{V}$. Therefore, $\Pr [\text{Game 0}] = 0$.

Game 1: This game is identical to **Game 0**, except that we use the knowledge extractor of π_2 to obtain witness $(p_\tau, open_p, ac_i, open_{ac_i}, ac_{i-1}, open_{ac_{i-1}}, v, \alpha)$. If extraction fails, we abort. The probability that extraction fails is negligible. Therefore $|\Pr [\text{Game 1}] - \Pr [\text{Game 0}]| = \nu_1$.

Game 2: This game is identical to **Game 1**, except that we abort when the extracted p_τ is not a valid price. The probability that $\mathcal{Z}$ distinguishes between **Game 2** and **Game 1** is bounded by the following theorem:

Theorem 4. *If the $(N + 1)$ -SDH assumption holds, then $|Pr[\mathbf{Game\ 2}] - Pr[\mathbf{Game\ 1}]| = \nu_2$.*

Proof. We build an algorithm $\mathcal{T}$ that, given an adversary $\mathcal{E}$ that outputs a valid proof π_2 for which an invalid price is extracted with non-negligible probability, breaks the weak unforgeability property of the signature scheme in [19] with non-negligible probability. Given $\mathcal{T}$, it is shown in [19] how to break the SDH assumption.

Upon receiving $(\text{init}, p_1, \dots, p_N)$ from $\mathcal{F}_{\text{POT}}$, $\mathcal{T}$ sends $(p_1, \dots, p_N)$ to the challenger and receives the public key pk and signatures on messages $(p_1, \dots, p_N)$ from the challenger. $\mathcal{T}$ employs these signatures to compute T . Upon receiving a request $Q = (V, D_p, D_i, \pi_2)$, $\mathcal{T}$ extracts from π_2 the witness $(p_\tau, \text{open}_p, ac_i, \text{open}_{ac_i}, ac_{i-1}, \text{open}_{ac_{i-1}}, v, \alpha)$. If p_τ is not in the list $(p_1, \dots, p_N)$, $\mathcal{T}$ outputs $(p_\tau, V^{1/v})$ as its forgery.

Game 3: This game is identical to **Game 2**, except that we abort when the extracted value ac_i does not equal $ac_{i-1} - p_\tau$. This means that $\mathcal{E}$ was able to break the binding property of Pedersen commitments, which holds under the discrete logarithm assumption. Therefore, the probability that $\mathcal{Z}$ distinguishes between **Game 3** and **Game 2** is bounded by the following theorem:

Theorem 5. *If the commitment scheme is binding, then $|Pr[\mathbf{Game\ 3}] - Pr[\mathbf{Game\ 2}]| = \nu_3$.*

Proof. We build an algorithm $\mathcal{T}$ that breaks the binding property of the commitment scheme with non-negligible probability. Given $\mathcal{T}$, it is shown in [23] how to break the discrete logarithm assumption.

Upon receiving the parameters par_{com} of the commitment scheme from the challenger, $\mathcal{T}$ uses them to compute the database commitment T . Upon receiving a request $Q = (V, D_p, D_i, \pi_2)$, $\mathcal{T}$ extracts $(p_\tau, \text{open}_p, ac'_i, \text{open}_{ac'_i}, v, \alpha)$ from π_2 . $\mathcal{T}$ knows $(ac_{i-1}, \text{open}_{ac_{i-1}})$ from extraction of the previous request. If ac'_i does not equal $ac_{i-1} - p_\tau$, $\mathcal{T}$, given commitment $D_i = \tilde{g}^{ac'_i} \tilde{h}^{\text{open}_{ac'_i}}$, is able to open D_i to $(ac_i, \text{open}_{ac_i})$, where $ac_i = ac_{i-1} - p_\tau$ and $\text{open}_{ac_i} = \text{open}_{ac_{i-1}} - \text{open}_p - \alpha$ (we recall that α fulfills this equality $\tilde{h}^\alpha = \tilde{g}^{ac_{i-1} - p_\tau - ac'_i} \tilde{h}^{\text{open}_{ac_{i-1}} - \text{open}_{p_\tau} - \text{open}_{ac_i}}$). Algorithm $\mathcal{T}$ outputs the tuple $(D_i, ac_i, \text{open}_{ac_i}, ac'_i, \text{open}'_{ac_i})$ to break the binding property of the commitment scheme.

Game 4: This game is identical to **Game 3**, except that we abort if $ac_i \notin [0, D_{\text{max}} = d^a)$. The probability that $\mathcal{Z}$ distinguishes between **Game 4** and **Game 3** is bounded by the following theorem:

Theorem 6. *If the $(d + 1)$ -SDH assumption holds, then $|Pr[\mathbf{Game\ 4}] - Pr[\mathbf{Game\ 3}]| = \nu_4$.*

Proof. We build an algorithm $\mathcal{T}$ that breaks the weak unforgeability property of the signature scheme in [19] with non-negligible probability. Given $\mathcal{T}$, it is shown in [19] how to break the SDH assumption.

Given $D_{max} = d^a$, when running $\text{InitVer}(1^\kappa, D_{max})$ $\mathcal{T}$ sends values $i \in \mathbb{Z}_d$ to the challenger and receives from the challenger the public key pk and signatures on values $i \in \mathbb{Z}_d$, and uses them to compute par_{ran} . Upon receiving a request, $\mathcal{T}$ gets the values $(\{V_j\}_{j \in \mathbb{Z}_a}, pok)$ of the range proof and extracts the witness $(open_\alpha, \{\alpha_j, v_j\}_{j \in \mathbb{Z}_a})$. (We recall that the range proof consists of $(\{V_j\}_{j \in \mathbb{Z}_a}, pok)$, where proof $pok = \text{NIPK}\{(open_\alpha, \{\alpha_j, v_j\}_{j \in \mathbb{Z}_a}) : C = h^{open_\alpha} \prod_{j \in \mathbb{Z}_a} (g^{u^j})^{\alpha_j} \wedge V_j = g^{v_j/(sk+\alpha_j)}\}$.) If $ac_i = \sum_{j \in \mathbb{Z}_a} \alpha_j d^j$ is not in $[0, D_{max})$, there must be a value $\alpha_j \notin \mathbb{Z}_d$. $\mathcal{T}$ outputs $(\alpha_j, V_j^{1/v_j})$ as its forgery.

Game 5: This game is identical to **Game 4**, except that now the database commitment T' is computed by using random messages. At this point π_3 is replaced by a simulated proof of a false statement. The probability that $\mathcal{Z}$ distinguishes between **Game 5** and **Game 4** is bounded by the following theorem:

Theorem 7. *If the $(N+1)$ -PDDH assumption holds, $|\Pr[\text{Game 5}] - \Pr[\text{Game 4}]| = \nu_5$.*

Proof. Given an adversary $\mathcal{E}$ which distinguishes between **Game 5** and **Game 4** with non-negligible probability, we construct an algorithm $\mathcal{T}$ that breaks the $(N+1)$ -PDDH assumption as follows. $\mathcal{T}$ receives the vector $(u, u^x, \dots, u^{x^{N+1}}, V) \in \mathbb{G}^{N+2} \times \mathbb{G}_t$, where $x \in \mathbb{Z}_p$, and the vector $(T_1, \dots, T_{N+1}) \in \mathbb{G}_t^{N+1}$. Let $T_0 = V$ and f be the polynomial defined as $f(X) = \prod_{i=1}^N (X + p_i) = \sum_{i=0}^N c_i X^i$. Then $\mathcal{T}$ sets $g \leftarrow u^{f(x)} = \prod_{i=0}^N (u^{x^i})^{c_i}$ and $y \leftarrow g^x = \prod_{i=0}^N (u^{x^{i+1}})^{c_i}$. If f_i is the polynomial defined by $f_i(X) = f(X)/(X + p_i) = \sum_{j=0}^{N-1} c_{i,j} X^j$, then $\mathcal{T}$ can also compute the values $A_i = g^{1/(x+p_i)}$ as $A_i \leftarrow \prod_{j=0}^{N-1} (u^{x^j})^{c_{i,j}}$. $\mathcal{T}$ then sets $H \leftarrow V^{f(x)} = \prod_{i=0}^N T_i^{c_i}$, and computes B_i as $H^{1/(x+p_i)}$ as $B_i \leftarrow \prod_{j=0}^{N-1} T_i^{c_{i,j}}$, and continues the simulation. One can check that if $T_i = V^{x^i}$ then the simulation corresponds to **Game 4**, while if $(T_1, \dots, T_{N+1})$ is random then the simulation corresponds to **Game 5**. Therefore, if $\mathcal{E}$ can distinguish between both, then $\mathcal{T}$ can break the $(N+1)$ -PDDH assumption.

$\mathcal{S}'_{\mathcal{B}}$ performs all the changes described in **Game 5**, and forwards and receives messages from $\mathcal{F}_{POT}$ as described in our simulation. The distribution produced in **Game 5** is identical to that of our simulation. Therefore, by summation we have that $|\Pr[\text{Game 5}]| \leq \nu_6$.

B Security Proofs for OFPOT Scheme

In order to prove Theorem 2, we need to build a simulator $\mathcal{S} = \{\mathcal{S}_{\mathcal{V}}, \mathcal{S}_{\mathcal{B}}\}$ that invokes a copy of adversary $\mathcal{E}$ and interacts with $\mathcal{F}_{\text{OFPOT}}$ and environment $\mathcal{Z}$ in

such a way that ensembles $\text{IDEAL}_{\mathcal{F}_{\text{OFFPOT}}, \mathcal{S}, \mathcal{Z}}$ and $\text{REAL}_{\text{OFFPOT}, \mathcal{E}, \mathcal{Z}}$ are computationally indistinguishable. Our simulator will use the simulators $\mathcal{S}' = \{\mathcal{S}'_{\mathcal{V}}, \mathcal{S}'_{\mathcal{B}}\}$ of the POT schemes, which are described in Appendix A and in Appendix C.3.

Simulation of buyer's security. In this case only the vendor $\mathcal{V}$ is corrupted.

- Upon receiving (crs) from $\mathcal{E}$, $\mathcal{S}_{\mathcal{V}}$ runs $\mathcal{S}'_{\mathcal{V}}$ on input (crs) to get (crs) and sends (crs, crs) to $\mathcal{E}$.
- Upon receiving $(pk_{\mathcal{V}}, T)$ from $\mathcal{E}$, $\mathcal{S}_{\mathcal{V}}$ runs $\mathcal{S}'_{\mathcal{V}}$ on input $(pk_{\mathcal{V}}, T)$ to get $(\text{init}, m_1, p_1, \dots, m_N, p_N)$ and sends $(\text{init}, m_1, p_1, \dots, m_N, p_N)$ to $\mathcal{F}_{\text{OFFPOT}}$.
- Upon receiving $(\text{deposit}, ac_0)$ from $\mathcal{F}_{\text{OFFPOT}}$, $\mathcal{S}_{\mathcal{V}}$ runs $\mathcal{S}'_{\mathcal{V}}$ on input $(\text{deposit}, ac_0)$ to get (P, D'_0) . $\mathcal{S}_{\mathcal{V}}$ runs $(sk_{\mathcal{B}}, pk_{\mathcal{B}}, \sigma_0) \leftarrow \text{OFInitB}(\text{crs}, D'_0)$ and sends $(P, pk_{\mathcal{B}}, \sigma_0)$ to $\mathcal{E}$.
- Upon receiving a request from $\mathcal{E}$, $\mathcal{S}_{\mathcal{V}}$ executes $(ask, apk) \leftarrow \text{OFInitA}(\text{crs})$ and sends apk to $\mathcal{E}$.
- Upon receiving $(pk_{\mathcal{V}}, pk_{\mathcal{B}})$ from $\mathcal{E}$, $\mathcal{S}_{\mathcal{V}}$ aborts if they do not equal $pk_{\mathcal{V}}$ previously sent by $\mathcal{E}$ or $pk_{\mathcal{B}}$ previously sent to $\mathcal{E}$.⁸
- Upon receiving $(\text{request}, i)$ from $\mathcal{F}_{\text{OFFPOT}}$, $\mathcal{S}_{\mathcal{V}}$ runs $\mathcal{S}'_{\mathcal{V}}$ on input (request) to obtain (Q_i, D'_i) . $\mathcal{S}_{\mathcal{V}}$ runs $\omega_i \leftarrow \text{OFReq}(sk_{\mathcal{B}}, apk, i, Q_i)$ and sends (i, Q_i, ω_i) to $\mathcal{E}$.
- Upon receiving R from $\mathcal{E}$, $\mathcal{S}_{\mathcal{V}}$ runs $\mathcal{S}'_{\mathcal{V}}$ on input R to obtain a bit b , and sends $(\text{response}, i, b)$ to $\mathcal{F}_{\text{OFFPOT}}$.
- Upon receiving $(\text{complete}, i, c)$ from $\mathcal{F}_{\text{OFFPOT}}$, if $c = 0$ $\mathcal{S}_{\mathcal{V}}$ sets $\sigma_i = \perp$. Otherwise $\mathcal{S}_{\mathcal{V}}$ runs $\sigma_i \leftarrow \text{OFComplete}(sk_{\mathcal{B}}, i, Q_i)$. $\mathcal{S}_{\mathcal{V}}$ sends σ_i to $\mathcal{E}$.
- Upon receiving a complain $((i, Q_i, \omega_i), (R), (i-1, Q_{i-1}, \sigma_{i-1}))$ ($((0, D_0, \sigma_0)$ when $i = 1$) from $\mathcal{E}$, $\mathcal{S}_{\mathcal{V}}$ parses Q_{i-1} to get D_{i-1} and runs $\text{POTVerReq}(\text{crs}, pk_{\mathcal{V}}, D_{i-1}, Q_i)$, $\text{OFVerReq}(pk_{\mathcal{B}}, apk, \omega_i, i, Q_i)$, $\text{OFVerComp}(pk_{\mathcal{B}}, \sigma_{i-1}, i-1, Q_{i-1})$ ($\text{OFInitV}(pk_{\mathcal{B}}, \sigma_0, D_0)$ when $i = 1$) and $\text{POTVerResp}(\text{crs}, pk_{\mathcal{V}}, R)$, and ignores the complaint if any of the algorithms outputs `reject`. If all the algorithms output `accept` and $\mathcal{E}$ did not receive before a signature on $H(i-1, Q_{i-1})$ or a verifiably encrypted signature on $H(i, Q_i)$, $\mathcal{S}_{\mathcal{V}}$ aborts. Otherwise, $\mathcal{S}_{\mathcal{V}}$ sends $(\text{complain}_{\mathcal{V}}, i)$ to $\mathcal{F}_{\text{OFFPOT}}$.
- Upon receiving $(\text{compInv}_{\mathcal{V}})$ from $\mathcal{F}_{\text{OFFPOT}}$, $\mathcal{S}_{\mathcal{V}}$ forwards $(\text{compInv}_{\mathcal{V}})$ to $\mathcal{E}$.
- Upon receiving $(\text{compSolved}_{\mathcal{V}})$ from $\mathcal{F}_{\text{OFFPOT}}$, $\mathcal{S}_{\mathcal{V}}$ executes $\sigma_i \leftarrow \text{OFComplete}(sk_{\mathcal{B}}, i, Q_i)$ and sends σ_i to $\mathcal{E}$.
- Upon receiving $(\text{complain}_{\mathcal{B}}, i)$ from $\mathcal{F}_{\text{OFFPOT}}$, $\mathcal{S}_{\mathcal{V}}$ sends $(\text{complain}_{\mathcal{B}}, i)$ to $\mathcal{E}$.
- Upon receiving a signature (j, Q_j, σ_j) from $\mathcal{E}$, $\mathcal{S}_{\mathcal{V}}$ proceeds as follows:
  - If the signature is invalid, $\mathcal{S}_{\mathcal{V}}$ sends $(\text{collaborate}, i, 0)$ to $\mathcal{F}_{\text{OFFPOT}}$.
  - If the signature is valid and $j \geq i$, $\mathcal{S}_{\mathcal{V}}$ aborts.
  - If it is valid and $j = i-1$, $\mathcal{S}_{\mathcal{V}}$ checks the request (i, Q'_i, ω'_i) (possibly) sent by $\mathcal{E}$ and aborts if it is correct but $\mathcal{E}$ was not given a verifiably encrypted signature on $H(i, Q'_i)$. Otherwise $\mathcal{S}_{\mathcal{V}}$ runs $\mathcal{S}'_{\mathcal{V}}$ on input (request) to obtain (Q_i, D'_i) . $\mathcal{S}_{\mathcal{V}}$ runs $\omega_i \leftarrow \text{OFReq}(sk_{\mathcal{B}}, apk, i, Q_i)$ and sends (i, Q_i, ω_i) to $\mathcal{E}$. Upon receiving R from $\mathcal{E}$, $\mathcal{S}_{\mathcal{V}}$ verifies the response. If it is not correct, $\mathcal{S}_{\mathcal{V}}$ sends $(\text{collaborate}, i, 0)$ to $\mathcal{F}_{\text{OFFPOT}}$, and otherwise $(\text{collaborate}, i, 1)$.

⁸ We recall that adjudicator $\mathcal{A}$ checks equality between the public keys sent by $\mathcal{V}$ and $\mathcal{B}$, and aborts if they are not equal.

- Upon receiving $(\text{guilty}_{\mathcal{V}})$ from $\mathcal{F}_{\text{OFFPOT}}$, $\mathcal{S}_{\mathcal{V}}$ forwards $(\text{guilty}_{\mathcal{V}})$ to $\mathcal{E}$.
- Upon receiving $(\text{compSolved}_{\mathcal{B}})$ from $\mathcal{F}_{\text{OFFPOT}}$, $\mathcal{S}_{\mathcal{V}}$ executes $\sigma_i \leftarrow \text{OFComplete}(sk_{\mathcal{B}}, i, Q_i)$. $\mathcal{S}_{\mathcal{V}}$ sends σ_i to $\mathcal{E}$.

Claim. When only $\mathcal{V}$ is corrupted, the distribution ensembles $\text{IDEAL}_{\mathcal{F}_{\text{OFFPOT}}, \mathcal{S}, \mathcal{Z}}$ and $\text{REAL}_{\text{OFFPOT}, \mathcal{E}, \mathcal{Z}}$ are computationally indistinguishable under the assumption that buyer's security in the POT scheme holds, under the unforgeability and opacity properties of the VES scheme, and under the assumption that H is collision-resistant.

We show by means of a series of hybrid games that the environment $\mathcal{Z}$ cannot distinguish between the real execution ensemble $\text{REAL}_{\text{OFFPOT}, \mathcal{E}, \mathcal{Z}}$ and the simulated ensemble $\text{IDEAL}_{\mathcal{F}_{\text{OFFPOT}}, \mathcal{S}, \mathcal{Z}}$ with non-negligible probability. We denote by $\Pr [\mathbf{Game} \ i]$ the probability that $\mathcal{Z}$ distinguishes between the ensemble of **Game** i and that of the real execution.

Game 0: This game corresponds to the execution of the real-world protocol with honest $\mathcal{B}$ and $\mathcal{A}$. Therefore, $\Pr [\mathbf{Game} \ 0] = 0$.

Game 1: This game performs all the changes described in $\mathcal{S}'_{\mathcal{V}}$. The security proof of the POT scheme demonstrates that $|\Pr [\mathbf{Game} \ 1] - \Pr [\mathbf{Game} \ 0]| = \nu_1$.

Game 2: This game proceeds as **Game 1**, except that the public keys $pk_{\mathcal{B}}$ and apk are replaced by other valid public keys $pk'_{\mathcal{B}}$ and apk' , and signatures ω and σ are computed on input $pk'_{\mathcal{B}}$ and apk' . Since $pk'_{\mathcal{B}}$ and apk' follow the same distribution as $pk_{\mathcal{B}}$ and apk , $|\Pr [\mathbf{Game} \ 2] - \Pr [\mathbf{Game} \ 1]| = 0$.

Game 3: This game proceeds as **Game 2**, except that we abort if $\mathcal{E}$ sends a verifiably encrypted signature ω_i on a pair (i, Q_i) and $\mathcal{E}$ did not receive before a verifiably encrypted signature on (i, Q_i) . The probability that environment $\mathcal{Z}$ distinguishes between **Game 3** and **Game 2** is bounded by the following theorem:

Theorem 8. *If the VES scheme is unforgeable and H is a collision resistant hash function, $|\Pr [\mathbf{Game} \ 3] - \Pr [\mathbf{Game} \ 2]| = \nu_2$.*

Proof. Given an adversary $\mathcal{E}$ that outputs ω_i on a new pair (i, Q_i) with non-negligible probability, we construct an algorithm $\mathcal{T}$ that either breaks the collision resistance property of H or wins the unforgeability game of the VES scheme (see [15]) with non-negligible probability. Upon receiving apk and pk from the challenger $\mathcal{C}_u$ of the unforgeability game and the hash function H from $\mathcal{C}_h$, $\mathcal{T}$ sets $pk_{\mathcal{B}} = (pk, H)$. Whenever $\mathcal{T}$ must simulate a request for $\mathcal{E}$ (when receiving $(\text{request}, i)$ from $\mathcal{F}_{\text{OFFPOT}}$ or in the complaint buyer phase), $\mathcal{T}$ computes values (i, Q_i) and sends $h_i = H(i, Q_i)$ to its VES creation oracle to obtain ω_i . Upon receiving $(\text{complete}, i, 1)$, $(\text{compSolved}_{\mathcal{V}})$ or $(\text{compSolved}_{\mathcal{B}})$ from $\mathcal{F}_{\text{OFFPOT}}$, $\mathcal{T}$ uses its adjudication oracle on input ω_i and $h_i = H(i, Q_i)$ to obtain σ_i . Eventually, $\mathcal{E}$ outputs (j, Q_j, ω_j) for new (j, Q_j) . $\mathcal{T}$ computes $h_j = H(j, Q_j)$. If h_j equals any of the previously computed h_i , $\mathcal{T}$ outputs $((j, Q_j), (i, Q_i))$ in order to break the collision resistance property of H . Otherwise $\mathcal{T}$ outputs (ω_j, h_j) as its forgery for the VES scheme.

Game 4: This game proceeds as **Game 3**, except that we abort if $\mathcal{E}$ sends a signature σ_i on a pair (i, Q_i) (or $(0, D_0)$) and $\mathcal{E}$ did not receive before a signature on (i, Q_i) . The probability that $\mathcal{Z}$ distinguishes between **Game 4** and **Game 3** is bounded by the following theorem:

Theorem 9. *If the VES scheme is opaque and H is a collision-resistant hash function, $|\Pr[\mathbf{Game\ 4}] - \Pr[\mathbf{Game\ 3}]| = \nu_3$.*

Proof. Given an adversary $\mathcal{E}$ that outputs σ_i on a new pair (i, Q_i) with non-negligible probability, we construct an algorithm $\mathcal{T}$ that either breaks the collision resistance property of H or wins the opacity game of the VES scheme (see [15]) with non-negligible probability. Upon receiving apk and pk from the challenger $\mathcal{C}_o$ of the opacity game and the hash function H from $\mathcal{C}_h$, $\mathcal{T}$ sets $pk_{\mathcal{B}} = (pk, H)$. Whenever $\mathcal{T}$ must simulate a request for $\mathcal{E}$ (when receiving $(\text{request}, i)$ from $\mathcal{F}_{\text{OFFPOT}}$ or in the complain buyer phase), $\mathcal{T}$ computes values (i, Q_i) and sends $h_i = H(i, Q_i)$ to its VES creation oracle to obtain ω_i . Upon receiving $(\text{complete}, i, 1)$, (compSolved_V) or $(\text{compSolved}_{\mathcal{B}})$ from $\mathcal{F}_{\text{OFFPOT}}$, $\mathcal{T}$ uses its adjudication oracle on input ω_i and $h_i = H(i, Q_i)$ to obtain σ_i . Eventually, $\mathcal{E}$ outputs (j, Q_j, σ_j) for new (j, Q_j) . $\mathcal{T}$ computes $h_j = H(j, Q_j)$. If h_j equals any of the previously computed h_i , $\mathcal{T}$ outputs $((j, Q_j), (i, Q_i))$ in order to break the collision resistance property of H . Otherwise $\mathcal{T}$ outputs (σ_j, h_j) to break the opacity property of the VES scheme.

$\mathcal{S}_V$ performs all the changes described in **Game 4**, and forwards and receives messages from $\mathcal{F}_{\text{OFFPOT}}$ as described in our simulation. The distribution produced in **Game 4** is identical to that of our simulation. Therefore, by summation we have that $\Pr[\mathbf{Game\ 4}] \leq \nu_4$.

Simulation of vendor's security. In this case only the buyer $\mathcal{B}$ is corrupted.

- Upon receiving (crs) from $\mathcal{E}$, $\mathcal{S}_{\mathcal{B}}$ runs $\mathcal{S}'_{\mathcal{B}}$ on input (crs) to get (crs) and sends (crs, crs) to $\mathcal{E}$.
- Upon receiving $(\text{init}, p_1, \dots, p_N)$ from $\mathcal{F}_{\text{OFFPOT}}$, $\mathcal{S}_{\mathcal{B}}$ executes $\mathcal{S}'_{\mathcal{B}}$ on input $(\text{init}, p_1, \dots, p_N)$ to obtain (pk_V, T) and sends them to $\mathcal{E}$.
- Upon receiving $(P, pk_{\mathcal{B}}, \sigma_0)$ from $\mathcal{E}$, $\mathcal{S}_{\mathcal{B}}$ runs $\mathcal{S}'_{\mathcal{B}}$ on input P to obtain (ac_0, D_0) . $\mathcal{S}_{\mathcal{B}}$ runs $\text{OFInitV}(pk_{\mathcal{B}}, \sigma_0, D_0)$ and ignores the request if the output is reject. Otherwise $\mathcal{S}_{\mathcal{B}}$ sends $(\text{deposit}, ac_0)$ to $\mathcal{F}_{\text{OFFPOT}}$.
- Upon receiving a request from $\mathcal{E}$, $\mathcal{S}_{\mathcal{B}}$ runs $(ask, apk) \leftarrow \text{OFInitA}(\text{crs})$ and sends apk to $\mathcal{E}$.
- Upon receiving $(pk_V, pk_{\mathcal{B}})$ from $\mathcal{E}$, $\mathcal{S}_{\mathcal{B}}$ aborts if they do not equal pk_V previously sent to $\mathcal{E}$ or $pk_{\mathcal{B}}$ previously sent by $\mathcal{E}$.
- Upon receiving (i, Q_i, ω_i) from $\mathcal{E}$, $\mathcal{S}_{\mathcal{B}}$ runs $\mathcal{S}'_{\mathcal{B}}$ on input Q_i to obtain (τ) . $\mathcal{S}_{\mathcal{B}}$ runs $\text{OFVerReq}(pk_{\mathcal{B}}, apk, \omega_i, i, Q_i)$ and, if the output is reject, $\mathcal{S}_{\mathcal{B}}$ ignores the request. Otherwise $\mathcal{S}_{\mathcal{B}}$ sends $(\text{request}, i, \tau)$ to $\mathcal{F}_{\text{OFFPOT}}$.
- Upon receiving $(\text{response}, i, m_{\tau})$ from $\mathcal{F}_{\text{OFFPOT}}$, $\mathcal{S}_{\mathcal{B}}$ executes $\mathcal{S}'_{\mathcal{B}}$ on input $(\text{response}, m_{\tau})$ to obtain a response (R) and sends (R) to $\mathcal{E}$.
- Upon receiving (σ_i) from $\mathcal{E}$, $\mathcal{S}_{\mathcal{B}}$ runs $\text{OFVerComp}(pk_{\mathcal{B}}, \sigma_i, i, Q_i)$ and, if the output is reject, sends $(\text{complete}, i, 0)$. Otherwise $\mathcal{S}_{\mathcal{B}}$ sends $(\text{complete}, i, 1)$.

- Upon receiving $(\text{compResp}_V, i, m_\tau)$ from $\mathcal{F}_{\text{OFFPOT}}$, $\mathcal{S}_B$ executes $\mathcal{S}'_B$ on input $(\text{response}, m_\tau)$ to obtain a response (R) and sends (R) to $\mathcal{E}$.
- Upon receiving a complain (i, Q_i, ω_i) from $\mathcal{E}$, $\mathcal{S}_B$ proceeds as for a request to obtain τ and finally sends $(\text{complain}_B, i, \tau)$ to $\mathcal{F}_{\text{OFFPOT}}$.
- Upon receiving (compInv_B) from $\mathcal{F}_{\text{OFFPOT}}$, $\mathcal{S}_B$ forwards (compInv_B) to $\mathcal{E}$.
- Upon receiving $(\text{compResp}_B, i, m_\tau)$ from $\mathcal{F}_{\text{OFFPOT}}$, $\mathcal{S}_B$ proceeds as for a response and finally sends R to $\mathcal{E}$.

Claim. When only $\mathcal{B}$ is corrupted, the distribution ensembles $\text{IDEAL}_{\mathcal{F}_{\text{OFFPOT}}, \mathcal{S}, \mathcal{Z}}$ and $\text{REAL}_{\text{OFFPOT}, \mathcal{E}, \mathcal{Z}}$ are computationally indistinguishable under the assumption that vendor's security in the POT scheme holds and under the extractability of the VES scheme.

We show that the real ensemble and the ideal ensemble are indistinguishable.

Game 0: This game corresponds to the execution of the real-world protocol with honest $\mathcal{B}$ and $\mathcal{A}$. Therefore, $\Pr [\text{Game 0}] = 0$.

Game 1: This game performs all the changes described in $\mathcal{S}'_B$. The security proof of the POT scheme demonstrates that $|\Pr [\text{Game 1}] - \Pr [\text{Game 0}]| = \nu_1$.

Game 2: This game proceeds as **Game 1**, except that the public key apk of the adjudicator is replaced by another valid public key computed via algorithm $\text{OFLInitA}(crs)$. Since both public keys follow the same distribution $|\Pr [\text{Game 2}] - \Pr [\text{Game 1}]| = 0$.

$\mathcal{S}_B$ performs all the changes described in **Game 2**, and forwards and receives messages from $\mathcal{F}_{\text{OFFPOT}}$ as described in our simulation. The distribution produced in **Game 2** is identical to that of our simulation. Therefore, by summation we have that $\Pr [\text{Game 2}] \leq \nu_1$.

C Universally Composable Priced Oblivious Transfer

We summarize here the POT scheme in [7], modifying the interface offered by the algorithms so that it can be used straightforwardly to instantiate our OFFPOT scheme.

C.1 Technical Preliminaries

Security Assumptions. Let $(p, \mathbb{G}, \mathbb{G}_t, e, g)$ be a pairing group setup. The l Hidden Strong Diffie-Hellman (HSDH) assumption [42] states that, on input $(g, g^\alpha, u) \in \mathbb{G}^3$ and a set of tuples $(g^{1/(\alpha+c_i)}, g^{c_i}, u^{c_i})_{i=1}^l$, it is hard to output a new tuple $(g^{1/(\alpha+c)}, g^c, u^c)$. The l Triple Diffie-Hellman (THD) assumption [43] states that, on input $(g, g^x, g^y) \in \mathbb{G}^3$ and a set of tuples $(g^{1/(x+c_i)}, c_i)_{i=1}^l$, it is hard to output $(g^{\mu x}, g^{\mu y}, g^{\mu xy})$ for $\mu \neq 0$. The Decision Linear (DLIN) assumption [44] states that, on input $(g, g^a, g^b, g^{ac}, g^{bd}, z) \in \mathbb{G}^6$ for random $a, b, c, d \in \mathbb{Z}_p$, it is hard to decide whether $z = g^{c+d}$.

Non-interactive Proofs of Knowledge. Let R be an efficiently computable relation and $L = \{y : \exists w | R(y, w) = \text{accept}\}$ be an NP-language. For tuples $(y, w) \in R$, we call y the instance and w the witness. A non-interactive proof of knowledge (NIPK) system [45] consists of algorithms (PKSetup, PKProve, PKVerify). Algorithm PKSetup(1^κ) outputs a common reference string par_{PK} . PKProve(par_{PK}, y, w) computes a proof pok of instance y by using witness w . PKVerify(par_{PK}, y, pok) outputs **accept** if pok is correct. A NIPK system should be complete, sound, and zero-knowledge (or witness-indistinguishable). See [46–49] for formal definitions.

A proof of knowledge needs to be extractable, which means that there exists a polynomial time extractor (PKExtSetup, PKExt). PKExtSetup(1^κ) generates parameters par_{PK} that are identically distributed to those generated by PKSetup and an extraction trapdoor td_{ext} . PKExt($par_{PK}, td_{ext}, y, pok$) extracts the witness w with all but negligible probability when PKVerify(par_{PK}, y, pok) outputs **accept**. In an f -extractable proof system [43], PKExt extracts a value z such that $\exists w : z = f(w) \wedge (y, w) \in R$. If $f(\cdot)$ is the identity function, we get the usual notion of extractability [45].

We consider NIPK about (unconditionally binding) commitments. ‘ x in C ’ denotes that there exists $open$ such that $C = \text{Com}(par_{com}, x, open)$. The following notation denotes an f -extractable NIPK for instance $(C_1, \dots, C_n, \text{Condition})$ with witness $(x_1, open_1, \dots, x_n, open_n, s)$ that allows extracting all the witness except the openings of the commitments (s is the part of the witness not related to the commitments in the instance):

$$\text{NIPK}\{ (x_1, \dots, x_n, s) : \text{Condition}(par_{PK}, x_1, \dots, x_n, s) \wedge \{x_i \text{ in } C_i\}_{i=1}^n \}$$

The POT scheme employs the Groth-Sahai NIPK system [50], which allows the computation of witness-indistinguishable proofs on statements about pairing product equations.⁹ The pairing group setup $(p, \mathbb{G}, \mathbb{G}_t, e, g)$ is part of the common reference string par_{PK} as output by PKSetup(1^κ) and the instance consists of the coefficients $\{a_q, b_q\}_{q=1 \dots Q} \in \mathbb{G}$, $t \in \mathbb{G}_t$, $\{\alpha_{q,i}, \beta_{q,i}\}_{q=1 \dots Q, i=1 \dots m} \in \mathbb{Z}_p$ of the pairing product equation: $\prod_{q=1}^Q e(a_q \prod_{i=1}^m x_i^{\alpha_{q,i}}, b_q \prod_{i=1}^m x_i^{\beta_{q,i}}) = t$. The prover knows $\{x_i\}_{i=1}^m$ that satisfy this equation.

Internally Groth-Sahai proofs prove relations between commitments. It is possible to add pre-existing Groth-Sahai commitments $\{C_i\}_{i=1 \dots n}$, $n \leq m$, to the instance for some of the x_i values. The corresponding openings $open_i$ become part of the witness. We can apply the aforementioned notation to Groth-Sahai proofs as follows:

$$\text{NIPK}\{ (\{x_i\}_{i=1}^n, \{x_i\}_{i=n+1}^m) : \prod_{q=1}^Q e(a_q \prod_{i=1}^n x_i^{\alpha_{q,i}} b_q \prod_{i=1}^m x_i^{\beta_{q,i}}) = t \wedge \{x_i \text{ in } C_i\}_{i=1}^n \}$$

We use the instantiation of GS proofs secure under the DLIN assumption.

⁹ Some classes of equations admit zero-knowledge proofs.

P-Signature Schemes. P-Signatures [43] are signature schemes equipped with a common reference string par_{sig} and a NIPK that allows proving possession of a signature on a committed message. The POT scheme employs the P-signature scheme (Setup, Kg, Sign, Vf) in [43]. Setup(1^κ) runs the GS PKSetup(1^κ) to get par_{PK} , picks random $u \in \mathbb{G}$ and outputs $par_{sig} = (par_{PK}, u)$. Kg(par_{sig}) picks a secret key $sk = (\alpha, \beta) \leftarrow \mathbb{Z}_p$ and computes a public key $pk = (v, w) = (g^\alpha, g^\beta)$. Sign(par_{sig}, sk, m) picks random $r \leftarrow \mathbb{Z}_p \setminus \{\frac{\alpha-m}{\beta}\}$ and outputs $\sigma = (\sigma_1, \sigma_2, \sigma_3) = (g^{1/(\alpha+m+\beta r)}, w^r, u^r)$. Vf(par_{sig}, pk, m, σ) outputs **accept** if $e(\sigma_1, v g^m \sigma_2) = e(g, g)$ and $e(u, \sigma_2) = e(\sigma_3, w)$. The NIPK is denoted as $NIPK\{(g^m, u^m, \sigma) : \text{Vf}(pk, \sigma, m) = \text{accept}\}$. This scheme is F -unforgeable¹⁰ ($F(m) = (g^m, u^m)$) under the HSDH and THD assumptions.

The POT scheme also employs the P-signature scheme for signing multi-block messages (Setup_n, Kg_n, Sign_n, Vf_n) in [7]. Setup_n(1^κ) runs the GS PKSetup(1^κ) to get par_{PK} , picks random $u \in \mathbb{G}$, and outputs $par_{sig} = (par_{PK}, u)$. Kg_n(par_{sig}) picks random $(\alpha, \beta_1, \dots, \beta_n, \lambda_1, \dots, \lambda_n) \leftarrow \mathbb{Z}_p$ and sets a public key $Pk = (v, g_1, \dots, g_n, u_1, \dots, u_n) = (g^\alpha, g^{\beta_1}, \dots, g^{\beta_n}, u^{\lambda_1}, \dots, u^{\lambda_n})$ and a secret key $Sk = (\alpha, \beta_1, \dots, \beta_n)$. Sign_n($par_{sig}, Sk, \mathbf{m}$) picks random $r \leftarrow \mathbb{Z}_p \setminus \{-(\alpha + \beta_1 m_1 + \dots + \beta_n m_n)\}$ and computes a signature $\sigma = (\sigma_1, \sigma_2, \sigma_3) = (g^{1/(\alpha+r+\beta_1 m_1+\dots+\beta_n m_n)}, g^r, u^r)$. Algorithm Vf_n($par_{sig}, Pk, \mathbf{m}, \sigma$) outputs **accept** if $e(\sigma_1, v \sigma_2 \prod_{i=1}^n g_i^{m_i}) = e(g, g)$ and $e(u, \sigma_2) = e(\sigma_3, g)$. This proof is denoted as $NIPK\{(\{g_i^{m_i}, u_i^{m_i}\}_{i=1}^n, \sigma) : \text{Vf}_n(Pk, \mathbf{m}, \sigma) = \text{accept}\}$. This scheme is F -unforgeable ($F(\mathbf{m}) = (g_1^m, u_1^m, \dots, g_n^m, u_n^m)$) under the HSDH and THD assumptions.

We recall that an F -unforgeable signature scheme can also be verified using the $F(m_i)$ values alone. We write, e.g., $\text{Vf}_n(Pk, \langle m_1, F(m_2), m_3 \rangle, \sigma)$ to indicate that the signature σ is verified using only the F value of message m_2 .

Non-Interactive Range Proof. The POT scheme employs the NIPK that a value $\alpha \in \mathbb{Z}_p$ lies in an interval $[0, A = d^a]$ proposed in [7], which applies to GS proofs the range proof in [27] (see Section 2). It consists of algorithms (InitVer, InitP, RProve). In the initialization phase, verifier runs InitVer(par_{sig}, A), which, on input par_{sig} output by Setup(1^κ), executes Kg(par_{sig}) to obtain (sk, pk) , computes $A_i = \text{Sign}(par_{sig}, sk, i)$ for all $i \in \mathbb{Z}_d$ and outputs $par_{ran} = (pk, \{A_i\}_{i \in \mathbb{Z}_d})$, and sends par_{ran} to the prover. The prover runs InitP(par_{sig}, par_{ran}), which outputs **accept** if, for all $i \in \mathbb{Z}_d$, Vf(par_{sig}, pk, i, A_i) outputs **accept**. Then the prover can compute multiple proofs. Algorithm RProve($par_{sig}, par_{ran}, \tilde{g}, \alpha, open_\alpha$) computes a NIPK that α committed in C_α lies in $[0, d^a]$: $NIPK\{(\tilde{g}^\alpha, \{g^{\alpha_j}, u^{\alpha_j}, A_{\alpha_j}\}_{j=0}^{a-1}) : \{\text{Vf}(pk, \alpha_j, A_{\alpha_j})\}_{j=0}^{a-1} \wedge e(g, \tilde{g}^\alpha) \prod_{j=0}^{a-1} e(\tilde{g}^{-d^j}, g^{\alpha_j}) = 1 \wedge \tilde{g}^\alpha \text{ in } C_\alpha\}$. This proof is abbreviated as $NIPK\{(\tilde{g}^\alpha) : 0 \leq \alpha < A \wedge \tilde{g}^\alpha \text{ in } C_\alpha\}$.

C.2 Construction

We recall the algorithms of the POT scheme. They are invoked in the same way we describe for our efficient POT scheme in Section 2.

¹⁰ We refer to [43] for the definition of F -unforgeability.

POTGenCRS($1^\kappa, p_{max}, D_{max}$) creates two GS reference strings $par_{PK}^\mathcal{V}$ and $par_{PK}^\mathcal{B}$ for the same setup $(p, \mathbb{G}, \mathbb{G}_t, e, g)$ (such that $-p_{max} > D_{max} \pmod{p}$), computes $(h_1, h_2, h_3) = (g^a, g^b, g^c)$ for random $a, b, c \leftarrow \mathbb{Z}_p$, picks random $u \leftarrow \mathbb{G}$ and outputs $crs = (par_{PK}^\mathcal{V}, par_{PK}^\mathcal{B}, u, h_1, h_2, h_3)$.¹¹

POTInitV($crs, m_1, p_1, \dots, m_N, p_N, D_{max}$) picks random $x_1, x_2 \leftarrow \mathbb{Z}_p$, sets $(w_1, w_2) = (h_3^{1/x_1}, h_3^{1/x_2})$ and runs **Kgn** to obtain (Pk, Sk) , where $Pk = (v, g_1, g_2, g_3, u_1, u_2, u_3)$ and $Sk = (\alpha, \beta_1, \beta_2, \beta_3)$. Then, for $i = 1$ to N , it picks random $r_1, r_2 \leftarrow \mathbb{Z}_p$, computes $(\sigma_1, \sigma_2, \sigma_3) \leftarrow \text{Sign}_n(par_{sig}, Sk, (r_1, r_2, p_i))$ and encrypts m_i as $C_i = (w_1^{r_1}, w_2^{r_2}, h_1^{r_1}, h_2^{r_2}, m_i \cdot h_3^{r_1+r_2}, g_1^{r_1}, g_2^{r_2}, u_1^{r_1}, u_2^{r_2}, \sigma_1, \sigma_2, \sigma_3, p_i)$. It runs **InitVer**(par_{sig}, D_{max}) to obtain par_{ran} , sets $pk_\mathcal{V} = (w_1, w_2, Pk, par_{ran})$, $sk_\mathcal{V} = (x_1, x_2)$ and $T = (C_1, \dots, C_N)$, and outputs $(T, sk_\mathcal{V}, pk_\mathcal{V})$.

POTInitB($crs, pk_\mathcal{V}, T, ac_0$) runs **InitP**(par_{sig}, par_{ran}) to verify par_{ran} . For $i = 1$ to N , it runs $\text{Vf}_n(Pk, \langle (c_6, c_8), (c_7, c_9), p_i \rangle, \sigma)$, where $\sigma = (\sigma_1, \sigma_2, \sigma_3)$. It checks that $e(c_1, h_1) = e(c_3, w_1)$, $e(c_2, h_2) = e(c_4, w_2)$, $e(h_1, c_6) = e(c_3, g_1)$, $e(h_2, c_7) = e(c_4, g_2)$. If all these checks hold, it picks random $(l_1, l_2) \leftarrow \mathbb{Z}_p$, runs and encrypts $ct = (c_1, c_2, c_3) = (w_1^{l_1}, w_2^{l_2}, ac_0 \cdot h_3^{l_1+l_2})$. It picks random $open_{ac_0}$, computes $D_0 = \text{Com}(par_{PK}^\mathcal{B}, g_3^{ac_0}, open_{ac_0})$ and outputs $P = (ct, open_{ac_0})$, $D'_0 = (ac_0, open_{ac_0}, D_0)$.¹² If not, it outputs reject.

POTGetDep($crs, sk_\mathcal{V}, P, D_{max}$) computes $ac_0 = c_3 / (c_1^{x_1} c_2^{x_2})$, checks that $ac_0 \in [0, A]$ and computes $D_0 = \text{Com}(par_{PK}^\mathcal{B}, g_3^{ac_0}, open_{ac_0})$. It outputs (D_0, ac_0) .

POTReq($crs, pk_\mathcal{V}, T, D'_{i-1}, \tau$) parses C_τ as $(c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, \sigma_1, \sigma_2, \sigma_3, p_\tau)$. It picks random $y_1, y_2 \leftarrow \mathbb{Z}_p$ and computes $(d_1, d_2) = (c_1 \cdot w_1^{y_1}, c_2 \cdot w_2^{y_2})$ and $(t_1, t_2) = (h_3^{y_1}, h_3^{y_2})$. It also picks a fresh $open_{ac_i}$ to compute $D_i = \text{Com}(g_3^{ac_i}, open_{ac_i})$, for $ac_i = ac_{i-1} - p_\tau$. It runs **PKProve** on input $par_{PK}^\mathcal{B}$ to compute a witness-indistinguishable proof pok_1 :

$$\begin{aligned} & \text{NIPK}\{(c_6, c_8, c_7, c_9, g_3^{p_\tau}, u_3^{p_\tau}, \sigma_1, \sigma_2, \sigma_3, g_3^{ac_i}, g_3^{ac_{i-1}}, c_1, c_2, t_1, t_2) : \\ & \quad \text{Vf}_n(Pk, \langle (c_6, c_8), (c_7, c_9), (g_3^{p_\tau}, u_3^{p_\tau}) \rangle, (\sigma_1, \sigma_2, \sigma_3)) = \text{accept} \wedge \\ & \quad e(w_1^{-1}, c_6) e(c_1, g_1) = 1 \wedge e(w_2^{-1}, c_7) e(c_2, g_2) = 1 \wedge \\ & \quad e(c_1, h_3) e(t_1, w_1) = e(d_1, h_3) \wedge e(c_2, h_3) e(t_2, w_2) = e(d_2, h_3) \wedge \\ & \quad e(g, g_3^{ac_{i-1}}) e(g^{-1}, g_3^{ac_i}) e(g^{-1}, g_3^{p_\tau}) = 1 \wedge \\ & \quad \wedge 0 \leq ac_i < A \wedge g_3^{ac_i} \text{ in } D_i \wedge g_3^{ac_{i-1}} \text{ in } D_{i-1}\} \end{aligned}$$

It outputs $Q = (d_1, d_2, pok_1, D_i)$, $Q' = (Q, \tau, y_1, y_2)$ and $D'_i = (ac_i, open_{ac_i}, D_i)$.

POTVerReq($crs, pk_\mathcal{V}, D_{i-1}, Q$) verifies pok_1 by running **PKVerify** on input $par_{PK}^\mathcal{B}$ and it outputs reject if it fails. For this verification, it uses the commitments D_{i-1} and D_i .

¹¹ par_{sig} is $(par_{PK}^\mathcal{V}, u)$ and is used for both the single message and multi-block message P-signature schemes.

¹² The original description uses $open_{ac_0} = 0$. We take a random value because these allows us to hide ac_0 from the adjudicator.

POTResp($crs, pk_{\mathcal{V}}, sk_{\mathcal{V}}, Q$) computes $(z_1, z_2) = (d_1^{x_1}, d_2^{x_2})$ and $z = z_1 \cdot z_2$ and runs PKProve on input $par_{PK}^{\mathcal{V}}$ to get a zero-knowledge proof of knowledge¹³ pok_2 :

$$\text{NIPK}\{(z_1, z_2) : e(z_1, w_1) = e(d_1, h_3) \wedge e(z_2, w_2) = e(d_2, h_3) \\ \wedge e(z_1, h_3)e(z_2, h_3) = e(z, h_3)\}$$

It outputs $R = (z, pok_2)$.

POTVerResp($crs, pk_{\mathcal{V}}, R$) verifies pok_2 by running PKVerify on input $par_{PK}^{\mathcal{V}}$. If verification fails, it outputs reject.

POTComplete(crs, T, R, Q') parses C_{τ} to obtain c_5 and outputs the message $m_{\tau} = c_5 / (z \cdot h_3^{-y_1} \cdot h_3^{-y_2})$.

C.3 Security Proof

Theorem 10. *This POT scheme securely realizes $\mathcal{F}_{POT}$ under the DL, HSDH and THD assumptions.*

We depict here the simulator $\mathcal{S}' = \{\mathcal{S}'_{\mathcal{V}}, \mathcal{S}'_{\mathcal{B}}\}$ given by [7] since it is used by the simulator $\mathcal{S}$ in the security proof of the OFPOT scheme, but we refer to [7] for the rest of the proof.

In the proof it is used the fact that Groth-Sahai proofs are partially extractable, composable witness-indistinguishable, and (given certain conditions) composable zero-knowledge. Algorithms (PKSimSetup, PKSimProve) are utilized to simulate proofs. PKSimSetup(1^{κ}) outputs an alternative setup par'_{PK} and a simulation trapdoor td_{sim} . PKSimProve(par'_{PK}, td_{sim}, y), on input simulation parameters par'_{PK} , outputs a proof pok for instance y such that PKVerify(par'_{PK}, y, pok) outputs accept.

Simulation of buyer's security. In this case only the vendor $\mathcal{V}$ is corrupted.

- $\mathcal{S}'_{\mathcal{V}}$ runs PKSetup to generate two Groth-Sahai reference strings $par_{PK}^{\mathcal{B}}$ and $par_{PK}^{\mathcal{V}}$ for the same pairing group setup $(p, \mathbb{G}, \mathbb{G}_t, e, g)$, where $-p_{max} > D_{max} \bmod p$ holds. $\mathcal{S}'_{\mathcal{V}}$ picks random $u \leftarrow \mathbb{G}$ and $a, b, c \leftarrow \mathbb{Z}_p$ and computes $h_3 = g^c$ and $(h_1, h_2) = (h_3^{1/a}, h_3^{1/b})$. $\mathcal{S}'_{\mathcal{V}}$ sets $crs = (par_{PK}^{\mathcal{V}}, par_{PK}^{\mathcal{B}}, u, h_1, h_2, h_3)$. When $\mathcal{F}_{CRS}$ is queried, $\mathcal{S}'_{\mathcal{V}}$ returns (crs) .
- Upon receiving $(pk_{\mathcal{V}}, T)$ from $\mathcal{E}$, $\mathcal{S}'_{\mathcal{V}}$ checks $(pk_{\mathcal{V}}, T)$ as depicted in POTInitB and aborts if it is not correct. Otherwise $\mathcal{S}'_{\mathcal{V}}$ parses T as $(C_1, \dots, C_N)$. For $i = 1, \dots, N$, $\mathcal{S}'_{\mathcal{V}}$ parses C_i to get (c_3, c_4, c_5, p_i) and sets $m_i = c_5 / (c_3^a c_4^b)$. $\mathcal{S}'_{\mathcal{V}}$ sends $(init, m_1, p_1, \dots, m_N, p_N)$ to $\mathcal{F}_{POT}$.
- Upon receiving $(deposit, ac_0)$ from $\mathcal{F}_{POT}$, $\mathcal{S}'_{\mathcal{V}}$ computes (P, D'_0) as explained in POTInitB. $\mathcal{S}'_{\mathcal{V}}$ sends (P) to $\mathcal{E}$ and keeps D'_0 .

¹³ To let this proof be zero-knowledge a new variable z_3 is introduced. The set of equations is $e(z_1, w_1)e(d_1^{-1}, z_3) = 1 \wedge e(z_2, w_2)e(d_2^{-1}, z_3) = 1 \wedge e(z_1 z_2, z_3)e(z^{-1}, z_3) = 1 \wedge e(w_1, z_3) = e(w_1, h_3)$.

- In the i th transfer phase, upon receiving (request) from $\mathcal{F}_{\text{POT}}$, $\mathcal{S}'_V$ executes $\text{POTReq}(crs, pk_V, T, D'_{i-1}, \tau_{min})$, where τ_{min} corresponds to the message with the lowest price, to obtain (Q, Q', D'_i) , and sends Q to $\mathcal{E}$.
- Upon receiving the response R from $\mathcal{E}$, $\mathcal{S}'_V$ runs $\text{POTVerResp}(crs, pk_V, R)$. If the output is **reject**, $\mathcal{S}'_V$ sends (response, 0) to $\mathcal{F}_{\text{POT}}$. Otherwise, $\mathcal{S}'_V$ sends (response, 1) to $\mathcal{F}_{\text{POT}}$ and keeps D'_i .

Simulation of vendor's security. In this case only the buyer $\mathcal{B}$ is corrupted.

- $\mathcal{S}'_B$ runs PKExtSetup to obtain par_{PK}^B and an extraction trapdoor td_{ext} , and algorithm PKSimSetup to obtain par_{PK}^V and a simulation trapdoor td_{sim} . Both use the same pairing group setup $(p, \mathbb{G}, \mathbb{G}_t, e, g)$, where $-p_{max} > D_{max} \bmod p$ holds. $\mathcal{S}'_B$ picks random $a, b, c \leftarrow \mathbb{Z}_p$ and computes $(h_1, h_2, h_3) = (g^a, g^b, g^c)$. $\mathcal{S}'_B$ picks random $u \leftarrow \mathbb{G}$ and sets $crs = (par_{\text{PK}}^V, par_{\text{PK}}^B, u, h_1, h_2, h_3)$. $\mathcal{S}'_B$ returns (crs) when $\mathcal{F}_{\text{CRS}}$ is queried.
- Upon receiving (init, $p_1, \dots, p_N$) from $\mathcal{F}_{\text{POT}}$, $\mathcal{S}'_B$ picks random messages $m'_1, \dots, m'_N \in \mathbb{G}_t^N$ and runs $\text{POTInitV}(crs, m'_1, p_1, \dots, m'_N, p_N, D_{max})$ to obtain (pk_V, T) . $\mathcal{S}'_B$ sends (pk_V, T) to $\mathcal{E}$.
- Upon receiving (P) , $\mathcal{S}'_B$ runs $(D_0, ac_0) \leftarrow \text{POTGetDep}(crs, sk_V, P, D_{max})$, sends (deposit, ac_0) to $\mathcal{F}_{\text{POT}}$ and keeps D_0 .
- In the i th transfer phase, upon receiving (Q) from $\mathcal{E}$, $\mathcal{S}'_B$ parses Q as (d_1, d_2, pok_1, D_i) . $\mathcal{S}'_B$ verifies the proof pok_1 by running PKVerify and utilizing (D_i, D_{i-1}) , and aborts if verification fails. Otherwise, $\mathcal{S}'_B$ executes $\text{PKExt}(par_{\text{PK}}^B, td_{ext}, pok_1)$ to extract the witness. Then, for $i = 1$ to N , $\mathcal{S}'_B$ compares the signature $(g_1^{r_1}, u_1^{r_1}, g_2^{r_2}, u_2^{r_2}, g_3^{p_i}, u_3^{p_i}, \sigma_1, \sigma_2, \sigma_3)$ in the witness with each of the signatures that are included in the ciphertexts that were sent to $\mathcal{E}$ in order to know the choice τ_i selected by $\mathcal{E}$. $\mathcal{S}'_B$ also compares the signatures $\{g^{\sigma_j}, u^{\sigma_j}, A_{\sigma_j}\}_{j=0}^{a-1}$ in the witness that correspond to the range proof with each of the signatures that were sent to $\mathcal{E}$ in par_{ran} . (This is done in order to ensure that $\mathcal{E}$ did not compute a forgery.) $\mathcal{S}'_B$ stores D_i and sends (request, τ_i) to $\mathcal{F}_{\text{POT}}$.
- $\mathcal{S}'_B$ receives either (response, m_{τ_i}) from $\mathcal{F}_{\text{POT}}$. If $m_{\tau_i} = \perp$, $\mathcal{S}'_B$ sends $(\perp)$ to $\mathcal{E}$. Otherwise $\mathcal{S}'_B$ uses the value of the ciphertext $c_5 = m'_i \cdot h_3^{r_1+r_2}$ and the values $(t_1, t_2) = (h_3^{y_1}, h_3^{y_2})$ in the extracted witness to compute a response $z = (c_5 t_1 t_2) / m_{\tau_i}$, and uses trapdoor td_{sim} to simulate proof pok_2 . $\mathcal{S}'_B$ sets $R = (z, pok_2)$ and sends (R) to $\mathcal{E}$.