Universally Composable Adaptive Priced Oblivious Transfer

Abstract

An adaptive k-out-of-N Priced Oblivious Transfer (POT) scheme is a two-party protocol between a vendor and a buyer. The vendor sells a set of messages $m_1, \ldots, m_N$ with prices $p_1, \ldots, p_N$. In each transfer phase $i = 1, \ldots, k$, the buyer chooses a selection value $\sigma_i \in \{1, \ldots, N\}$ and interacts with the vendor to buy message m_{σ_i} in such a way that the vendor does not learn σ_i and the buyer does not get any information about the other messages.

We present a POT scheme secure under pairing-related assumptions in the standard model. Our scheme is universally composable and thus, unlike previous results, preserves security when it is executed with multiple protocol instances that run concurrently in an adversarially controlled way. Furthermore, after an initialization phase of complexity $O(N)$, each transfer phase is optimal in terms of rounds of communication and it has constant computational and communication cost. To achieve these properties, we design the first efficient non-interactive proof of knowledge that a value lies in a given interval we are aware of.

Keywords: Universally composable security, priced oblivious transfer, bilinear maps, non-interactive range proofs of knowledge.

1 Introduction

A number of studies [31] show that transaction security and privacy concerns are among the main reasons that discourage the use of e-commerce. Although sometimes it is argued that users who claim to be worried about their privacy do not consistently take actions to protect it, recent research [41] demonstrates that, when they are confronted to a prominent display of private information, they not only prefer vendors that offer better privacy protection but also are willing to pay higher prices to purchase from more privacy protective websites. Therefore, it is of interest for vendors to deploy e-commerce applications where buyers need to disclose the minimum information needed to carry out their transactions.

So far, the solutions proposed to develop privacy-enhancing e-commerce of digital goods can roughly be divided into two categories: those that hide the identity of the buyer from the vendor (anonymous purchase), and those that hide which goods are bought (oblivious purchase). Anonymous purchase [27, 33] usually employs anonymous e-cash [17, 12, 3] to construct systems where buyers can withdraw coins from a bank and spend them without revealing their identity. These systems have several shortcomings. First, they hinder customer management (e.g. the vendor cannot easily apply marketing techniques like giving discounts to regular buyers). Second, they do not allow for other methods of payment. Finally, strong anonymity is difficult to achieve and there exist several attacks to reduce it [5].

Oblivious purchase is thus more appealing in scenarios where full anonymity cannot be obtained or when the disadvantages that anonymity causes are important. Oblivious purchase permits effective customer management and allows for every method of payment. Like for anonymous purchase [27, 33], it has also been shown how to integrate it into existing Digital Rights Management systems [35]. One can argue that, since the vendor does not know which items are sold, he can find it difficult to discover which products are more demanded. However, we note that this information can be obtained from other sources, e.g., by conducting marketing researches.

Oblivious purchase employs the Priced Oblivious Transfer (POT) [1] primitive, which is a generalization of the well-known Oblivious Transfer (OT) [37] primitive intended to permit private purchases. OT is a two-party protocol between a sender $\mathcal{S}$ and a receiver $\mathcal{R}$, where $\mathcal{S}$ offers a set of messages $m_1, \ldots, m_N$ to $\mathcal{R}$. $\mathcal{R}$ chooses selection values $\sigma_1, \ldots, \sigma_k \in \{1, \ldots, N\}$ and interacts with $\mathcal{S}$ in such a way that $\mathcal{R}$ learns $m_{\sigma_1}, \ldots, m_{\sigma_k}$ and nothing about the other messages, and $\mathcal{S}$ does not learn anything about $\sigma_1, \ldots, \sigma_k$.

POT is a two-party protocol between a vendor $\mathcal{V}$ and a buyer $\mathcal{B}$, where $\mathcal{V}$ sells a set of messages $m_1, \ldots, m_N$ with prices $p_1, \ldots, p_N$ to $\mathcal{B}$. Besides the requirements that $\mathcal{V}$ must not learn $\sigma_1, \ldots, \sigma_k$ and $\mathcal{B}$ must not learn anything about the other messages, in POT $\mathcal{B}$ must pay prices $p_{\sigma_1}, \ldots, p_{\sigma_k}$ without $\mathcal{V}$ learning anything about the amount of money paid.

Both OT and POT admit an adaptive variant [36] ($OT^N_{k\times 1}$,$POT^N_{k\times 1}$) where, in transfer phase i, $\mathcal{R}$ or $\mathcal{B}$ may choose σ_i after receiving $m_{\sigma_{i-1}}$. The adaptive variant is more suitable for constructing an oblivious database, enabling applications of OT such as medical record storage or location-based services [36, 32], and the deployment of privacy-preserving e-commerce.

Previous work. The universally composable security paradigm [15] provides a framework for representing cryptographic protocols and analyzing their security. Protocols that are proven UC-secure maintain their security even when they are run concurrently with an unbounded number of arbitrary protocol instances controlled by an adversary.

Traditionally, security in OT was analyzed under a half-simulation model, where simulation security is required against $\mathcal{R}$, but just stand-alone privacy is required against $\mathcal{S}$. This notion was showed to admit practical attacks against receiver's security [36]. [13], as well as subsequent works [25], present efficient adaptive OT schemes in a full-simulation model. However, these works are not UC-secure because they use black-box simulation with adversarial rewinding in their security proofs.

Recently, an adaptive UC-secure OT scheme was proposed [26]. They utilize the approach of assisted decryption used in [13, 25], where $\mathcal{S}$ sends to $\mathcal{R}$ a collection of ciphertexts and in each transfer phase helps $\mathcal{R}$ to decrypt one of them. As pointed out in [26], this approach allows for transfer phases with constant computational and communication complexity, and it is suitable to ensure that $\mathcal{S}$ does not change the messages in each transfer phase, which are important properties for constructing an oblivious database. This is in contrast to the approach used in other non-adaptive UC-secure OT schemes [20, 42], where, in each transfer phase, $\mathcal{R}$ hands a set of keys to $\mathcal{S}$, who sends back a collection of ciphertexts such that $\mathcal{R}$ is able to decrypt only one of them.

Despite this recent progress in OT, so far there are not efficient POT schemes whose security is proven within the UC security paradigm. The first POT scheme [1], as well as subsequent works [40], analyze security in the half-simulation model. [20] explains why these protocols fail even under sequential composition and shows a practical attack.

The existing conditional oblivious transfer schemes [19, 6], where sender with input x and receiver with input y interact in such a way that a transfer is completed only when $q(x, y) = 1$ for some public predicate $q(\cdot, \cdot)$, are non-adaptive and employ the half-simulation model. On the other hand, security of both the non-adaptive [30, 39] and the adaptive [29] Generalized Oblivious Transfer schemes proposed so far, which can be instantiated as non-adaptive and adaptive POT schemes respectively, depends on the underlying OT scheme utilized to implement them, but we note that these solutions are rather inefficient. Finally, access control schemes for OT based on stateful anonymous credentials [18] are not UC-secure either.

Our contribution. We present a $POT^N_{k\times 1}$ scheme that is UC-secure under the assumption that there is an honestly generated common reference string. Security is proven in a static corruption model without relying on random oracles. After an initialization phase of complexity $O(N)$, each transfer phase is optimal in terms of rounds of communication and has constant computational and communication cost.

Our construction follows the approach in [1] of building a prepaid mechanism where $\mathcal{B}$ makes an initial deposit to $\mathcal{V}$. In each transfer phase, $\mathcal{B}$ chooses a selection value σ_i, proves that she has enough funds to buy message m_{σ_i} and subtracts price p_{σ_i} from her deposit, while $\mathcal{V}$ learns neither p_{σ_i} nor the new value of the deposit. For this purpose, $\mathcal{B}$ employs a zero-knowledge proof of knowledge that she updates her account correctly and that the new account is non-negative. To allow for the latter we design a non-interactive range proof of knowledge by applying the efficient interactive range proof recently proposed in [11] to the non-interactive proof system due to Groth and Sahai [28]. This is the first efficient non-interactive proof of knowledge in the standard model to prove that a value lies in a given interval we are aware of.

We also employ the assisted decryption approach and some techniques utilized in the adaptive UC-secure OT scheme in [26]. Specifically, we use double trapdoor encryption and we prove security of ciphertexts under the DLIN [9] assumption. Nonetheless, unlike [26], we make extensive use of P-signatures [4], i.e., signature schemes that have efficient non-interactive proofs of signature possession, to let $\mathcal{B}$ prove that she computes her requests honestly. In particular, we employ a slightly modified variant of the P-signature scheme for signing blocks of messages proposed in [3], which is secure under the HSDH [10] and TDH [4] assumptions. (P-signatures also utilize Groth-Sahai proofs, which we instantiate using the DLIN assumption.) The use of multi-block P-signatures allows our scheme to have a smaller ciphertext size than the one in [26]. We note that our POT scheme can easily be simplified to obtain an OT scheme, which constitutes an alternative to the one in [26].

Outline of the paper. In Section 2 we briefly review the universally composable security paradigm and we define the ideal functionality for POT. The security assumptions we use, the Groth-Sahai proof system, and other cryptographic building blocks are described in Section 3. In Section 4 we show how to construct a non-interactive range proof. Finally, in Section 5 we depict the multi-block P-signature scheme and our POT scheme.

2 Definitions

Adaptive k-out-of-N priced oblivious transfer ($POT_{k\times1}^{N}$). A $POT_{k\times1}^{N}$ scheme is a two-party protocol between a vendor $\mathcal{V}$ and a buyer $\mathcal{B}$. In the initialization phase, $\mathcal{V}$ receives messages $(m_1, \ldots, m_N)$ with prices $(p_1, \ldots, p_N)$ as input. $\mathcal{B}$ receives an initial deposit ac_0 as input. $\mathcal{B}$ stores state information B_0 and $\mathcal{V}$ stores state information V_0 and outputs ac_0. After that, $\mathcal{V}$ and $\mathcal{B}$ engage in up to k transfer phases. In the ith transfer, $\mathcal{V}$ gets state information V_{i-1} as input, and $\mathcal{B}$ gets state information B_{i-1} and selection value $\sigma_i \in \{1, \ldots, N\}$. If $ac_0 - \sum_{j\in S} p_{\sigma_j} \geq 0$, where S contains the indices of all transfers that ended successfully, then $\mathcal{V}$ stores state information V_i and $\mathcal{B}$ stores state information B_i and outputs m_{σ_i}. Otherwise $\mathcal{V}$ stores $V_i = V_{i-1}$ and $\mathcal{B}$ stores $B_i = B_{i-1}$.

Universally composable security. We employ the universally composable security framework [15] with static corruptions to prove security of our construction. In this framework, parties are modeled as probabilistic polynomial time interactive Turing machines. A protocol ψ is UC-secure if there exists no environment $\mathcal{Z}$ that can distinguish whether it is interacting with adversary $\mathcal{A}$ and parties running protocol ψ or with the ideal process for carrying out the desired task, where ideal adversary $\mathcal{E}$ and dummy parties interact with an ideal functionality $\mathcal{F}_\psi$. More formally, we say that protocol ψ emulates the ideal process when, for all environments $\mathcal{Z}$, the ensembles $IDEAL_{\mathcal{F}_\psi,\mathcal{E},\mathcal{Z}}$ and $REAL_{\psi,\mathcal{A},\mathcal{Z}}$ are computationally indistinguishable. We refer to Appendix A for a more detailed description of the UC framework.

Our construction operates in the $\mathcal{F}_{CRS}$-hybrid plain model, where parties have access to an honestly-generated common reference string crs and to authenticated channels. As in [26], we assume that $\mathcal{Z}$ obtains crs from $\mathcal{A}$. This allows the simulator $\mathcal{E}$ to set up a crs with trapdoor information to be able to simulate $\mathcal{A}$ in the security proof.

Below we recall the description of the ideal functionality for generating common reference strings ($\mathcal{F}_{CRS}$) [16]. $\mathcal{F}_{CRS}$ is parameterized with a distribution D and a set of participants $\mathcal{P}$, which is restricted to contain the buyer $\mathcal{B}$ and the vendor $\mathcal{V}$ of the POT scheme only. We also describe an ideal functionality for POT ($\mathcal{F}_{POT}$) based on the ideal functionality for OT given in [26].

$\mathcal{F}_{CRS}$**.** On input (sid, 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 (sid, crs, r) to P.

$\mathcal{F}_{POT}$**.** Parameterized with integers (N, l), a maximum price p_{max}, and a deposit upper bound S, and running with a vendor $\mathcal{V}$ and a buyer $\mathcal{B}$, $\mathcal{F}_{POT}$ works as follows:

- On input a message $(sid, vendor, m_1, \ldots, m_N, p_1, \ldots, p_N)$ from $\mathcal{V}$, where each $m_i \in \{0,1\}^l$ and each $p_i \in [0, p_{max}]$, it stores $(m_1, \ldots, m_N)$ and $(p_1, \ldots, p_N)$ and sends $(sid, p_1, \ldots, p_N)$ to $\mathcal{B}$ and to the adversary.
- On input a message $(sid, buyer, deposit)$, where $deposit \in [0, \ldots, S)$, if a $(sid, vendor, \ldots)$ message was not received before, then it does nothing. Otherwise, it stores $deposit$ and sends $(sid, deposit)$ to $\mathcal{V}$.
- On input a message $(sid, buyer, \sigma)$ from $\mathcal{B}$, where $\sigma \in \{1, \ldots, N\}$, if either messages $(sid, vendor, m_1, \ldots, m_N, p_1, \ldots, p_N)$ and $(sid, buyer, deposit)$ were not received before or $deposit - p_\sigma < 0$, then it does nothing. Otherwise, it sends $(sid, request)$ to $\mathcal{V}$ and receives (sid, b) in response. It hands (sid, b) to the adversary. If $b = 0$, it sends $(sid, \perp)$ to $\mathcal{B}$. If $b = 1$, it updates $deposit = deposit - p_\sigma$ and sends (sid, m_σ) to $\mathcal{B}$.

3 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 the following properties:

(a) *Bilinearity*. A map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is bilinear if $e(a^x, b^y) = e(a, b)^{xy}$;

(b) *Non-degeneracy*. For all generators $g \in \mathbb{G}$, $e(g, g)$ generates $\mathbb{G}_T$;

(c) *Efficiency*. There exists an efficient algorithm 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 $a, b \in \mathbb{G}$.

3.1 Assumptions

The security of our scheme relies on the Hidden Strong DH assumption [10], the Triple DH assumption [4], and the Decision Linear assumption [9]:

Definition 1 (HSDH) *On input $(g, g^{\alpha}) \in \mathbb{G}^2$, $u \in \mathbb{G}$, and a set of tuples $\{g^{1/(\alpha+c_i)}, g^{c_i}, u^{c_i}\}_{i=1}^{l}$, the l-HSDH assumption holds if it is computationally hard to output a new tuple $(g^{1/(\alpha+c)}, g^c, u^c)$.*

Definition 2 (TDH) *On input $(g, g^x, g^y) \in \mathbb{G}^3$ and a set of tuples $\{c_i, g^{1/(x+c_i)}\}_{i=1}^{l}$, the l-TDH assumption holds if it is computationally hard to output a tuple $(g^{\mu x}, g^{\mu y}, g^{\mu xy})$ for $\mu \in \mathbb{Z}_p/\{0\}$.*

Definition 3 (DLIN) *On input $(g, g^a, g^b, g^{ac}, g^{bd}, z) \in \mathbb{G}^6$ for random exponents $a, b, c, d \in \mathbb{Z}_p$, the DLIN assumption holds if it is computationally hard to decide whether $z = g^{c+d}$.*

3.2 Non-interactive Zero-knowledge 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 system [38] consists of algorithms $\mathsf{PKSetup}$, $\mathsf{PKProve}$ and $\mathsf{PKVerify}$. $\mathsf{PKSetup}(1^{\kappa})$ outputs a common reference string crs_{PK}. $\mathsf{PKProve}(crs_{PK}, y, w)$ computes a proof pok of instance y by using witness w. $\mathsf{PKVerify}(crs_{PK}, y, pok)$ outputs accept if pok is correct.

Zero-knowledge captures the notion that a verifier learns nothing from the proof but the truth of the statement. Witness indistinguishability is a weaker property that guarantees that the verifier learns nothing about which witness was used in the proof. In either case, we will also require soundness, meaning that an adversarial prover cannot convince an honest verifier of a false statement, and completeness, meaning that all correctly computed proofs are accepted by the honest verification algorithm. See [23, 22, 7, 21] for formal definitions.

In addition, a proof of knowledge needs to be extractable. Extractability means that there exists a polynomial time extractor $(\mathsf{PKExtractSetup}, \mathsf{PKExtract})$. $\mathsf{PKExtractSetup}(1^{\kappa})$ generates parameters crs_{PK} that are identically distributed to the ones generated by algorithm $\mathsf{PKSetup}$ and an extraction trapdoor td_{ext}. $\mathsf{PKExtract}(crs_{PK}, td_{ext}, y, pok)$ extracts the witness w with all but negligible probability when $\mathsf{PKVerify}(crs_{PK}, y, pok)$ outputs accept.

We recall the notion of *f-extractability* defined by Belenkiy et al. [4], which is an extension of the original definition of extractability (as given by De Santis et al. [38]). In an f-extractable proof system the extractor $\mathsf{PKExtract}$ extracts a value z such that $\exists w : z = f(w) \wedge R(y, w) = \text{accept}$. If $f(\cdot)$ is the identity function, we get the usual notion of extractability.

Commitment schemes. A non-interactive commitment scheme consists of the algorithms $\mathsf{ComSetup}$ and Commit. $\mathsf{ComSetup}(1^{\kappa})$ generates the parameters of the commitment scheme $params_{Com}$. Commit $(params_{Com}, x, open)$ outputs a commitment C to x using auxiliary information $open$. A commitment is opened by revealing $(x, open)$ and checking $\mathsf{Commit}(params_{Com}, x, open) = C$. 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 $params_{Com}$.)

A notation for f-extractable non-interactive proofs of knowledge. We are interested in NIPK about (unconditionally binding) commitments. By '$x \,\text{in}\, C$' we denote that there exists $open$ such that $C = \mathsf{Commit}(params_{Com}, x, open)$. Following Camenisch and Stadler [14] and Belenkiy et at. [4], we use the following notation to express an f-extractable NIPK for instance $(C_1, \ldots, C_n, \mathsf{Condition})$ with witness $(x_1, open_1, \ldots, x_n, open_n, s)$ that allows to extract all the witness except the openings of the commitments:

$$\pi \leftarrow \mathsf{NIPK}\{\ (x_1, \ldots, x_n, s\ ) : \mathsf{Condition}(crs, x_1, \ldots, x_n, s) \land x_1 \,\mathrm{in}\, C_1 \land \ldots \land x_n \,\mathrm{in}\, C_n\}$$

The f-extractability of a NIPK ensures that, with overwhelming probability over the choice of crs, if PKVerify accepts then we can extract $(x_1, \ldots, x_n, s)$ from π, such that x_i is the content of the commitment C_i, and $\mathsf{Condition}(crs, x_1, \ldots, x_n, s)$ is satisfied. To further abbreviate this notation, we omit crs when it is clear from the context.

Applying the notation to Groth-Sahai proofs. Groth-Sahai proofs [28] allow proving statements about pairing product equations. The pairing group setup $(p, \mathbb{G}, \mathbb{G}_T, e, g)$ is part of the common reference string crs_{PK} as output by $\mathsf{PKSetup}(1^\kappa)$ and the instance consists of the coefficients $\{a_q\}_{q=1\ldots Q}$, $\{b_q\}_{q=1\ldots Q} \in \mathbb{G}$, $t \in \mathbb{G}_T$, $\{\alpha_{q,i}\}_{q=1\ldots Q, i=1\ldots m}$, $\{\beta_{q,i}\}_{q=1\ldots Q, i=1\ldots m} \in \mathbb{Z}_p$ 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. A homomorphism guarantees that the same relations also hold for the committed values. Normally, as the first step in creating the proof, the prover prepares commitments $\{C_i\}_{i=1\ldots m}$ for all values x_i in $\mathbb{G}$. Then, the instance, known to the prover and the verifier, is the pairing product equation alone (i.e., its coefficients).

Additionally, it is possible to add pre-existing Groth-Sahai commitments $\{C_i\}_{i=1\ldots n}$, $n \leq m$, to the instance for some of the x_i values. The corresponding openings $open_i$ become part of the witness. The proof will be computed in the same way, except that for values with existing commitments no fresh commitments need to be computed. We will write $C_i \leftarrow \mathsf{Commit}(x_i, open_i)$ to create Groth-Sahai commitments. Note that they use parameters contained in the crs_{PK} of the Groth-Sahai proof system. The Groth-Sahai proof system generates f-extractable witness indistinguishable[1] NIPK of the form:

$$\mathsf{NIPK}\{(x_1, \ldots, x_n, x_{n+1}, \ldots x_m) : \prod_{q=1}^{Q} e(a_q \prod_{i=1}^{n} x_i^{\alpha_{q,i}}, b_q \prod_{i=1}^{m} x_i^{\beta_{q,m}}) = t \land x_1 \,\mathrm{in}\, C_1 \land \cdots \land x_n \,\mathrm{in}\, C_n\}$$

3.3 P-Signature Schemes

A signature scheme consists of the algorithms Keygen, Sign and VerifySig. Keygen outputs a secret key sk and a public key pk. $\mathsf{Sign}(sk, m)$ outputs a signature s of message m. $\mathsf{VerifySig}(pk, m, s)$ outputs accept if s is a valid signature of m and reject otherwise. (This definition can be extended to support multi-block messages $\vec{m} = \{m_1, \ldots, m_n\}$.) A signature scheme must be correct and unforgeable [24]. Informally speaking, correctness implies that the VerifySig algorithm always accepts an honestly generated signature. Unforgeability means that no p.p.t adversary should be able to output a message-signature pair (s, m) unless he has previously obtained a signature on m.

P-Signatures are defined by Belenkiy et al. in [4] as signature schemes equipped with a common reference string crs_{Sig} and a NIPK that allows proving possession of a signature of a committed message. Belenkiy et al. show how to use the Groth-Sahai proof system to build this proof. Since in their constructions $m \in \mathbb{Z}_p$ and Groth-Sahai proofs prove knowledge of a witness in $\mathbb{G}$, they need to compute a bijection $F(m) \in \mathbb{G}$ and prove knowledge of $F(m)$. To avoid that given a secure signature scheme an adversary may still be able to compute a forgery $(s, F(m))$ even though he is unable to compute (s, m), [4] defines F-unforgeability, which means that no p.p.t adversary can output $(s, F(m))$ without previously obtaining a signature on m.

[1] Some classes of pairing product equations also admit zero-knowledge proofs.

4 Non-Interactive Range Proof

We construct an efficient non-interactive range proof that a committed value $\sigma \in \mathbb{Z}_p$ lies in an interval $[0..A)$. Our scheme is based on the efficient interactive range proof recently proposed in [11]. The technique of [11] consists in writing σ in base-d to show that it lies in an interval $[0..d^a)$. First, the verifier sends the prover signatures S_i on d-ary digits, i.e., $i \in \mathbb{Z}_d$. Then the prover proves that $\sigma = \sum_{j \in \mathbb{Z}_a} \sigma_j d^j$ and that all σ_j are d-ary digits. For the latter, she proves possession of a verifier's signature on σ_j. Our idea consists in employing P-signatures, which allow for a non-interactive proof of signature possession, to construct a non-interactive range proof following this approach.

A handy P-signature scheme. We employ the P-signature scheme of [4] that is based on the strong Boneh-Boyen signature scheme [8].

$\mathsf{Setup}(1^\kappa)$ runs the Groth-Sahai $\mathsf{PKSetup}(1^\kappa)$ to obtain crs_{PK} for pairing groups $(p, \mathbb{G}, \mathbb{G}_T, e, g)$, picks random $u \in \mathbb{G}$, and outputs $crs_{Sig} = (crs_{PK}, u)$.

$\mathsf{Keygen}(crs_{Sig})$ picks a secret key $sk = (\alpha, \beta) \leftarrow \mathbb{Z}_p$ and computes a public key $pk = (v, w) = (g^\alpha, g^\beta)$.

$\mathsf{Sign}(crs_{Sig}, sk, m)$ chooses $r \leftarrow \mathbb{Z}_p / \{\frac{\alpha - msg}{\beta}\}$ and sets $s = (s_1, s_2, s_3) = (g^{1/(\alpha + m + \beta r)}, w^r, u^r)$.

$\mathsf{VerifySig}(crs_{Sig}, pk, m, s)$ outputs accept if $e(s_1, vg^m s_2) = e(g, g)$, $e(u, s_2) = e(s_3, w)$. Otherwise, it outputs reject.

Using Groth-Sahai proofs, [4] shows how to construct a NIPK of such a signature. This is a proof of a pairing product equation of the form

$$\mathsf{NIPK}\{(g^m, u^m, s_1, s_2, s_3) : e(s_1, vg^m s_2) = e(g, g) \wedge e(u, s_2) = e(s_3, w) \wedge e(u, g^m) = e(u^m, g)\}$$

We will abbreviate this expression by writing $\mathsf{NIPK}\{(g^m, u^m, s) : \mathsf{VerifySig}(pk, s, m) = \text{accept}\}$. This scheme is F-unforgeable ($F(m) = (g^m, u^m)$) under the HSDH and TDH assumptions.

Range proof. The range proof uses a common reference string crs_{Sig} as produced by Setup. In addition, we require that the verifier can distribute public parameters $params_{Range} \leftarrow \mathsf{RPInitVerifier}(crs_{Sig}, A)$. These parameters do not need to be honestly generated, as they can be verified by the prover using RPInitProver.

$\mathsf{RPInitVerifier}(crs_{Sig}, A)$. On input $A = d^a$, it runs $\mathsf{Keygen}(crs_{Sig})$ to obtain (sk, pk), and, $\forall i \in \mathbb{Z}_d$, it computes $S_i = \mathsf{Sign}(crs_{Sig}, sk, i)$. It outputs $params_{Range} = (pk, \{S_i\}_{i \in \mathbb{Z}_d})$.

$\mathsf{RPInitProver}(crs_{Sig}, params_{Range})$. It parses $params_{Range}$ to obtain pk and $\{S_i\}_{i \in \mathbb{Z}_d}$. It verifies the signatures by computing, for all $i \in \mathbb{Z}_d$, $\mathsf{VerifySig}(crs_{Sig}, pk, i, S_i)$. If these verifications succeed, it outputs accept. Otherwise it outputs reject.

$\mathsf{RangeProve}(crs_{Sig}, params_{Range}, \tilde{g}, \sigma, open_\sigma)$ computes the following proof for a commitment $C_\sigma = \mathsf{Commit}(\tilde{g}^\sigma, open_\sigma)$:

$$\begin{aligned}
&\mathsf{NIPK}\{(\tilde{g}^\sigma, \{g^{\sigma_j}, u^{\sigma_j}, S_{\sigma_j}\}_{j=0}^{a-1}) : \\
&\quad \{\mathsf{VerifySig}(pk, \sigma_j, S_{\sigma_j}))\}_{j=0}^{a-1} \wedge && (1)\\
&\quad e(g, \tilde{g}^\sigma) \prod_{j=0}^{a-1} e(\tilde{g}^{-d^j}, g^{\sigma_j}) = 1 \wedge \tilde{g}^\sigma \text{ in } C_\sigma\} \; . && (2)
\end{aligned}$$

Intuitively, (1) ensures that each σ_j is a d-ary digit by proving that the value was used by the verifier to compute a signature S_{σ_j}, and (2) proves that σ is correctly decomposed, i.e., that $\sigma = \sum_{j\in\mathbb{Z}_a} \sigma_j d^j$. We use the short form $\mathsf{NIPK}\{(\tilde{g}^\sigma) : 0 \leq \sigma < A \wedge \tilde{g}^\sigma \,\mathrm{in}\, C_\sigma\}$ to refer to this proof.

This proof is only witness indistinguishable. While this is sufficient for our application, it is possible to make the proof zero-knowledge using techniques described in [28]. This proof can be extended to handle intervals of the form $[A..B)$ in the same way as in [11].

5 A UC-Secure Adaptive k-out-of-N Priced Oblivious Transfer Scheme

5.1 Intuition Behind our Construction

Our priced oblivious transfer scheme is based on the oblivious transfer scheme presented in [26]. Specifically, it is an assisted decryption scheme that employs double trapdoor encryption (based on the linear encryption scheme in [9]). The ciphertext of message m contains values $(w_1^{r_1}, w_2^{r_2}, h_1^{r_1}, h_2^{r_2}, m \cdot h_3^{r_1+r_2})$, where (w_1, w_2) are public parameters generated by vendor $\mathcal{V}$ and (h_1, h_2, h_3) belong to the common reference string. $(w_1^{r_1}, w_2^{r_2})$ are used by buyer $\mathcal{B}$ to generate the request message in each transfer phase, whereas $(h_1^{r_1}, h_2^{r_2})$ are used in the security proof by the ideal protocol adversary $\mathcal{E}$ to obtain the messages from $\mathcal{V}$ without the necessity of extracting a secret key from a proof of knowledge. This is useful because if the secret key is a value in $\mathbb{Z}_p$, then Groth-Sahai proofs do not permit its extraction. In order to be able to decrypt, $\mathcal{E}$ creates trapdoor information when generating the crs. (We note that the environment learns crs through the adversary. As mentioned in [26], there are impossibility results for realizing UC-secure OT if $\mathcal{E}$ cannot craft crs.) In addition, by using double trapdoor encryption we also prove the security of ciphertexts under the DLIN assumption.

The message space is $\{0,1\}^l$, but we abuse notation and also write m to denote the corresponding group element in $\mathbb{G}$ according to some efficient and invertible mapping. We will do the same when encrypting the account ac_0 that is a value in $\mathbb{Z}_p$ using linear encryption. For such a mappings between a bit string $\{0,1\}^l$ and an element in $\mathbb{G}$ see, e.g., [2].

The ciphertexts also contain signatures of $(w_1^{r_1}, w_2^{r_2})$ that are used to ensure that $\mathcal{B}$ generates her requests honestly. [26] employs signature schemes that sign elements in $\mathbb{G}$. However, we use a multi-block P-signature scheme that signs elements in $\mathbb{Z}_p$, and thus we sign values (r_1, r_2). Consequently, we need to provide $\mathcal{B}$ with the values $F(r_1, r_2) = (g_1^{r_1}, g_2^{r_2}, u_1^{r_1}, u_2^{r_2})$ of this signature scheme. Nonetheless, we note that in our scheme the ciphertexts have less group elements than in [26].

In order to permit oblivious purchases, our $POT_{k\times 1}^N$ extends the $OT_{k\times 1}^N$ construction sketched above. We follow the approach of [1] of building a prepaid scheme, where in the initialization phase the buyer $\mathcal{B}$ pays an initial deposit ac_0 to the vendor $\mathcal{V}$, and in subsequent transfer phases this deposit is subtracted by the price p_σ of the message that is being bought.

The POT scheme must ensure that $\mathcal{V}$ learns neither the price of the message nor the new value of the account, but also that $\mathcal{B}$ pays the right price for the message and that she has enough funds to buy it. To achieve this, in the initialization phase $\mathcal{B}$ sends a commitment to the deposit. In the ith transfer, $\mathcal{B}$ sends a commitment to the new value of the account ac_i and proves that (1) this value is correct, i.e., that $ac_i = ac_{i-1} - p_\sigma$, and that (2) it is non-negative. In order to allow for (1), we need to ensure that $\mathcal{B}$ uses the right price. To accomplish this, $\mathcal{V}$ adds the price of the message to the message block (r_1, r_2, p_σ). Thanks to that, when $\mathcal{B}$ proves possession of the signature, $\mathcal{B}$ can include in this proof a pairing product equation to prove that $ac_i = ac_{i-1} - p_\sigma$. To verify this proof, $\mathcal{V}$ employs the commitment to ac_{i-1} that he got in the previous transfer phase. To achieve (2), in the initialization phase $\mathcal{V}$ computes parameters of the range proof and hands then to $\mathcal{B}$. In each transfer phase, $\mathcal{B}$ proves that the new value of the account ac_i belongs to $[0..A)$, where A is the deposit upper bound.

5.2 P-Signatures for Blocks of Messages

We describe an F-unforgeable P-signature scheme for signing multiple message blocks that is based on the single block scheme presented in [3]. Let $\vec{m} = \langle m_1, \dots, m_n \rangle$ denote n message blocks.

$\mathsf{Setup_n}(1^\kappa)$ runs the Groth-Sahai $\mathsf{PKSetup}(1^\kappa)$ to obtain crs_{PK} for pairing groups $(p, \mathbb{G}, \mathbb{G}_T, e, g)$, picks random $u \in \mathbb{G}$, and outputs $crs_{Sig} = (crs_{PK}, u)$.

$\mathsf{Keygen_n}(crs_{Sig})$ 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 $Sk = (\alpha, \beta_1, \dots, \beta_n)$.

$\mathsf{Sign_n}(crs_{Sig}, Sk, \vec{m})$ chooses random $r \leftarrow \mathbb{Z}_p/\{-(\alpha + \beta_1 m_1 + \ldots + \beta_n m_n)\}$ and computes a signature $s = (s_1, s_2, s_3) = (g^{1/(\alpha + r + \beta_1 m_1 + \ldots + \beta_n m_n)}, g^r, u^r)$.

$\mathsf{VerifySig_n}(crs_{Sig}, Pk, \vec{m}, s)$ outputs accept if $e(s_1, v s_2 \prod_{i=1}^n g_i^{m_i}) = e(g, g)$ and $e(u, s_2) = e(s_3, g)$.

We extend the multi-block signature scheme with a protocol for proving possession of a signature.

$$\mathsf{NIPK}\{(\{g_i^{m_i}, u_i^{m_i}\}_{i=1}^n, s_1, s_2, s_3) : \{e(u_i, g_i^{m_i}) e(u_i^{m_i}, g_i^{-1}) = 1\}_{i=1}^n \wedge$$
$$e(u, s_2) e(s_3, g^{-1}) = 1 \wedge e(s_1, v s_2 \prod_{i=1}^n g_i^{m_i}) = e(g, g)\}$$

We use the short form $\mathsf{NIPK}\{(\{g_i^{m_i}, u_i^{m_i}\}_{i=1}^n, s) : \mathsf{VerifySig_n}(Pk, \vec{m}, s) = \text{accept}\}$ to refer to this proof.

Theorem 1 *Let $F(m_1, \dots, m_n) = (g_1^{m_1}, u_1^{m_1}, \dots, g_n^{m_n}, u_n^{m_n})$. This signature scheme is F-unforgeable under the HSDH and TDH assumptions.* We prove Theorem 1 in Appendix B.

We make use of the observation that an F-unforgeable signature scheme can also be verified using the $F(m_i)$ values alone, i.e., without knowing m_i. Like in the proof, an additional check of the equations $\{e(u_i, g_i^{m_i}) e(u_i^{m_i}, g_i^{-1}) = 1\}_{i=1}^n$ is needed to verify that the $F(m_i)$ values are constructed correctly. Moreover, the $F(m_i)$ values are sufficient to create a proof of possession of a signature. We write, e.g., $\mathsf{VerifySig_n}(Pk, \langle m_1, F(m_2), m_3 \rangle, s)$ to indicate that the signature s is verified using only the F value of message m_2.

5.3 Construction

We begin with a high level description of the priced oblivious transfer scheme. The vendor $\mathcal{V}$ and the buyer $\mathcal{B}$ interact in the initialization phase and in several transfer phases. Details on the algorithms can be found below. We recall that the scheme is parameterized with integers (N, l) for the number of messages and their length, an upper bound p_{max} for the prices and an upper bound $A = d^a$ for the deposit.

Initialization phase. On input $(sid, vendor, m_1, \dots, m_N, p_1, \dots, p_N)$ for the vendor and $(sid, buyer, ac_0)$ for the buyer (that fulfill the restrictions imposed by the parameters of the scheme):

1. $\mathcal{V}$ queries $\mathcal{F}_{CRS}$ with $(sid, \mathcal{V}, \mathcal{B})$. $\mathcal{F}_{CRS}$ runs $\mathsf{POTGenCRS}(1^\kappa, p_{max}, A)$ and sends (sid, crs) to $\mathcal{V}$.
2. $\mathcal{B}$ queries $\mathcal{F}_{CRS}$ with $(sid, \mathcal{V}, \mathcal{B})$. $\mathcal{F}_{CRS}$ sends (sid, crs) to $\mathcal{B}$.
3. $\mathcal{V}$ runs $\mathsf{POTInitVendor}(crs, m_1, \dots, m_N, p_1, \dots, p_N, A)$ to compute a database commitment T and a secret key sk, and sends (sid, T) to $\mathcal{B}$.
4. $\mathcal{B}$ gets (sid, T) and runs $(P, D_0^{(priv)}) \leftarrow \mathsf{POTInitBuyer}(crs, T, ac_0)$. $\mathcal{B}$ aborts if the output is reject. Otherwise, $\mathcal{B}$ sends (sid, P) to $\mathcal{V}$. ($\mathcal{B}$ also needs to pay an amount of ac_0 to $\mathcal{V}$ through an arbitrary payment channel.)

5. (Upon receiving the money) $\mathcal{V}$ runs $(D_0, ac_0) \leftarrow \mathsf{POTGetDeposit}(crs, P, A)$ and checks that ac_0 corresponds to the amount of money received.

$\mathcal{V}$ stores state information $V_0 = (T, sk, D_0)$ and outputs (sid, ac_0), and $\mathcal{B}$ stores state information $B_0 = (T, D_0^{(priv)})$.

Transfer phase. In the ith transfer, $\mathcal{V}$ with state information V_{i-1} and input $(sid, vendor, b)$ and $\mathcal{B}$ with state information B_{i-1} and input $(sid, buyer, \sigma_i)$ interact as follows:

1. $\mathcal{B}$ runs $\mathsf{POTRequest}(crs, T, D_{i-1}^{(priv)}, \sigma_i)$ to obtain a request Q and private state $(Q^{(priv)}, D_i^{(priv)})$. $\mathcal{B}$ sends (sid, Q) to $\mathcal{V}$ and stores $(sid, Q^{(priv)}, D_i^{(priv)})$.
2. $\mathcal{V}$ gets (sid, Q). If $b = 0$, $\mathcal{V}$ sends $(sid, \perp)$ to $\mathcal{B}$. Otherwise $\mathcal{V}$ runs $\mathsf{POTRespond}(crs, T, sk, D_{i-1}, Q)$ to obtain a response R and state D_i. $\mathcal{V}$ sends (sid, R) to $\mathcal{B}$.
3. $\mathcal{B}$ receives (sid, R) and runs $\mathsf{POTComplete}(crs, T, R, Q^{(priv)})$ to obtain m_{σ_i}.

$\mathcal{V}$ stores state information $V_i = (T, sk, D_i)$, and $\mathcal{B}$ stores state information $B_i = (T, D_i^{(priv)})$ and outputs (sid, m_{σ_i}).

$\mathsf{POTGenCRS}(1^\kappa, p_{max}, A)$**.** Given security parameter κ, it generates two Groth-Sahai reference strings $crs_{PK}^{\mathcal{V}}$ and $crs_{PK}^{\mathcal{B}}$ for the same pairing group setup $(p, \mathbb{G}, \mathbb{G}_T, e, g)$ such that $-p_{max} > A \mod p$ holds. (In the proof of security the two setups allow the simulator to simultaneously make use of knowledge extraction and simulation for the first and the second proof respectively.) It picks random $a, b, c \leftarrow \mathbb{Z}_p$ and computes $(h_1, h_2, h_3) = (g^a, g^b, g^c)$. It picks random $u \leftarrow \mathbb{G}$. It outputs $crs = (crs_{PK}^{\mathcal{V}}, crs_{PK}^{\mathcal{B}}, u, h_1, h_2, h_3)$.[2]

$\mathsf{POTInitVendor}(crs, m_1, \ldots, m_N, p_1, \ldots, p_N, A)$**.** On input the messages $(m_1, \ldots, m_N)$ and the prices $(p_1, \ldots, p_N)$:

1. It parses crs to obtain $crs_{Sig} = (crs_{PK}^{\mathcal{B}}, u)$ and (h_1, h_2, h_3).
2. It picks random $x_1, x_2 \leftarrow \mathbb{Z}_p$ and sets $(w_1, w_2) = (h_3^{1/x_1}, h_3^{1/x_2})$.
3. It runs $\mathsf{Keygen_n}$ to obtain (Pk, Sk), where $Pk = (v, g_1, g_2, g_3, u_1, u_2, u_3)$ and $Sk = (\alpha, \beta_1, \beta_2, \beta_3)$.
4. For $i = 1, \ldots, N$, it encrypts m as follows:
   (a) It picks random $r_1, r_2 \leftarrow \mathbb{Z}_p$.
   (b) It computes $(s_1, s_2, s_3) = \mathsf{Sign_n}(crs_{Sig}, Sk, (r_1, r_2, p_i))$.
   (c) It sets $C_i = (w_1^{r_1}, w_2^{r_2}, h_1^{r_1}, h_2^{r_2}, m_i \cdot h_3^{r_1+r_2}, g_1^{r_1}, g_2^{r_2}, u_1^{r_1}, u_2^{r_2}, s_1, s_2, s_3, p_i)$.
5. $\mathcal{V}$ runs $\mathsf{RPInitVerifier}(crs_{Sig}, A)$ to obtain $params_{Range}$.
6. It sets $pk = (w_1, w_2, Pk, params_{Range})$, $sk = (x_1, x_2)$ and $T = (pk, C_1, \ldots, C_N)$. It outputs (T, sk).

$\mathsf{POTInitBuyer}(crs, T, ac_0)$**.** On input a database commitment T and a deposit $ac_0 \in [0..A)$:

1. It parses crs to obtain $crs_{Sig} = (crs_{PK}^{\mathcal{B}}, u)$, T as $(pk, C_1, \ldots, C_N)$, pk as $(w_1, w_2, Pk, params_{Range})$ and the public key Pk as $(v, g_1, g_2, g_3, u_1, u_2, u_3)$.
2. It runs $\mathsf{RPInitProver}(crs_{Sig}, params_{Range})$ to verify $params_{Range}$.

[2]Note that the set $crs_{Sig} = (crs_{PK}^{\mathcal{B}}, u)$ is used as common reference string for both the multi-block signature scheme and the single-message signature scheme, which is used for running the range proof.

3. For $i = 1, \ldots, N$:
   (a) It parses $C_i = (c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, s_1, s_2, s_3, p_i)$.
   (b) It runs $\mathsf{VerifySig_n}(Pk, \langle (c_6, c_8), (c_7, c_9), p_i \rangle, s)$.
   (c) It verifies that $e(c_1, h_1) = e(c_3, w_1) \wedge e(c_2, h_2) = e(c_4, w_2) \wedge e(h_1, c_6) = e(c_3, g_1) \wedge e(h_2, c_7) = e(c_4, g_2)$.
4. If not all these checks verify, it outputs reject. Otherwise it picks random $(r_1, r_2) \leftarrow \mathbb{Z}_p$ and sets $P = (w_1^{r_1}, w_2^{r_2}, ac_0 \cdot h_3^{r_1+r_2})$ and $D_0^{(priv)} = (ac_0, open_{ac_0} = 0)$. It outputs $(P, D_0^{(priv)})$.

$\mathsf{POTGetDeposit}(crs, P, A)$. It works as follows:

1. It parses P as $(c_1, c_2, c_3, c_4, c_5)$.
2. It computes $ac_0 = c_5/(c_1^{x_1} c_2^{x_2})$ and checks that $ac_0 \in [0..A)$.
3. It sets $D_0 = \mathsf{Commit}(g_3^{ac_0}, 0)$. It outputs (D_0, ac_0).

$\mathsf{POTRequest}(crs, T, D_{i-1}^{(priv)}, \sigma)$. On input a database commitment T and a selection value $\sigma \in \{1, \ldots, N\}$, it works as follows:

1. It parses T as $(pk, C_1, \ldots, C_N)$, pk as $(w_1, w_2, Pk, params_{Range})$, crs to obtain $(crs_{PK}^{\mathcal{B}}, u, h_3)$ and C_σ as $(c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, s_1, s_2, s_3, p_\sigma)$.
2. 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 $(t_1, t_2) = (h_3^{y_1}, h_3^{y_2})$.
3. It parses $D_{i-1}^{(priv)}$ as $(ac_{i-1}, open_{ac_{i-1}})$ to compute $D_{i-1} = \mathsf{Commit}(g_3^{ac_{i-1}}, open_{ac_{i-1}})$. It also picks a fresh $open_{ac_i}$ to compute $D_i = \mathsf{Commit}(g_3^{ac_i}, open_{ac_i})$, for $ac_i = ac_{i-1} - p_\sigma$.
4. It runs PKProve on input $crs_{PK}^{\mathcal{B}}$ to compute a witness-indistinguishable proof pok_1:

$$\begin{aligned}
&\mathsf{NIPK}\{(c_6, c_8, c_7, c_9, g_3^{p_\sigma}, u_3^{p_\sigma}, s_1, s_2, s_3, g_3^{ac_i}, g_3^{ac_{i-1}}, c_1, c_2, t_1, t_2) : \\
&\quad \mathsf{VerifySig_n}(Pk, \langle (c_6, c_8), (c_7, c_9), (g_3^{p_\sigma}, u_3^{p_\sigma}) \rangle, (s_1, s_2, s_3)) = \text{accept} \wedge \\
&\quad e(w_1^{-1}, c_6) e(c_1, g_1) = 1 \wedge e(w_2^{-1}, c_7) e(c_2, g_2) = 1 \wedge \\
&\quad e(c_1, h_3) e(t_1, w_1) = e(d_1, h_3) \wedge e(c_2, h_3) e(t_2, w_2) = e(d_2, h_3) \wedge \\
&\quad e(g, g_3^{ac_{i-1}}) e(g^{-1}, g_3^{ac_i}) e(g^{-1}, g_3^{p_\sigma}) = 1 \wedge \\
&\quad \wedge\, 0 \leq ac_i < A \wedge g_3^{ac_i} \text{ in } D_i \wedge g_3^{ac_{i-1}} \text{ in } D_{i-1}\}
\end{aligned}$$

5. It sets $Q = (d_1, d_2, pok_1, D_i)$, $Q^{(priv)} = (Q, \sigma, y_1, y_2)$ and $D_i^{(priv)} = (ac_i, open_{ac_i})$. It outputs $(Q, Q^{(priv)}, D_i^{(priv)})$.

$\mathsf{POTRespond}(crs, T, sk, D_{i-1}, Q)$. On input a database commitment T, a secret key sk, private state D_{i-1}, and a request Q, it works as follows:

1. It parses crs to obtain $(crs_{PK}^{\mathcal{V}}, crs_{PK}^{\mathcal{B}}, u, h_3)$, T as $(pk, C_1, \ldots, C_N)$, pk as $(w_1, w_2, Pk, params_{Range})$, sk as (x_1, x_2), Q as (d_1, d_2, pok_1, D_i).
2. It verifies pok_1 by running PKVerify on input $crs_{PK}^{\mathcal{B}}$ and it aborts if the output is reject. For this verification, it uses the commitments D_{i-1} and D_i.
3. It computes $(z_1, z_2) = (d_1^{x_1}, d_2^{x_2})$ and $z = z_1 \cdot z_2$.
4. It runs PKProve on input $crs_{PK}^{\mathcal{V}}$ to compute a zero-knowledge proof of knowledge[3] pok_2:

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

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

	POT scheme	OT scheme [26]	Our underlying OT scheme
crs	23	16	23
Database T	$12N + 3d + 11$	$18N + 11$	$12N + 7$
Request	$86 + 30a$	66	65
Response	28	35	28

Table 1: Performance comparison with the OT scheme in [26]

5. It outputs $R = (z, pok_2)$ and D_i.

$\mathsf{POTComplete}(crs, T, R, Q^{(priv)})$. On input a database commitment T, a response R and private state $Q^{(priv)}$:

1. It parses crs to obtain $(crs^{\mathcal{V}}_{PK}, h_3)$, T as $(pk, C_1, \ldots, C_N)$, R as (z, pok_2) and $Q^{(priv)}$ as (Q, σ, y_1, y_2).
2. It verifies pok_2 by running $\mathsf{PKVerify}$ on input $crs^{\mathcal{V}}_{PK}$. If verification fails, it outputs reject.
3. It parses C_σ to obtain c_5 and it outputs the message $m_\sigma = c_5/(z \cdot h_3^{-y_1} \cdot h_3^{-y_2})$.

Theorem 2 *This POT scheme securely realizes $\mathcal{F}_{POT}$.* We prove Theorem 2 in Appendix C.

5.4 Properties and Extensions

This scheme offers extra features over previous ones [1]. Namely, it permits that several messages have the same price without scaling up prices and accounts, and it allows the vendor to charge different prices for the same message to different buyers, which can be used to apply marketing techniques like making discounts to regular or underage buyers. This can be done by recomputing the signatures included in the ciphertexts on different prices depending on the particular buyer. In order to allow for a precomputed database, $\mathcal{V}$ can assign buyers to ℓ different groups and associate to each group $j \in \{1, \ldots, \ell\}$ a different price for each message m_i by signing $s^{(j)} = \mathsf{Sign_n}(crs_{Sig}, Sk, (r_1, r_2, j, p_{ij}))$. (Note that r_1 and r_2 have the same value in the signatures of all the groups in order to reuse the same encryption of m_i.) In the transfer phase, when proving possession of the multi-block P-signature $s^{(j)}$ for their group, buyers must reveal the attribute j.

The POT scheme can be simplified to obtain an OT scheme, which constitutes an alternative to the one in [26]. Additionally, the multi-block signature scheme provides high flexibility to implement other access control policies for oblivious transfer beyond those required for POT. For example, if an index i is signed instead of price p_i, then access control methods based on stateful anonymous credentials [18], which support a wide variety of policies, can be applied.

5.5 Efficiency Analysis and Comparison

In Table 1 we compare the performance of our POT scheme with the performance of the OT scheme in [26] and with the OT scheme obtained by simplifying our POT scheme. We show the number of group elements in the crs, in the database T, in the request message, and in the response message. (We recall that the deposit upper bound is $A = d^a$.) See Appendix D for more details.

References

[1] W. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. In B. Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, p. 119–135. Springer, 2001.

[2] G. Ateniese, J. Camenisch, and B. de Medeiros. Untraceable rfid tags via insubvertible encryption. In V. Atluri, C. Meadows, and A. Juels, editors, *ACM Conference on Computer and Communications Security*, p. 92–101. ACM, 2005.

[3] M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya. Compact e-cash and simulatable vrfs revisited, 2008.

[4] M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya. P-signatures and noninteractive anonymous credentials. In R. Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, p. 356–374. Springer, 2008.

[5] O. Berthold, H. Federrath, and M. Köhntopp. Project "anonymity and unobservability in the internet". In *CFP '00: Proceedings of the tenth conference on Computers, freedom and privacy*, p. 57–65, New York, NY, USA, 2000. ACM.

[6] I. F. Blake and V. Kolesnikov. Strong conditional oblivious transfer and computing on intervals. In *In Advances in Cryptology - ASIACRYPT 2004*, p. 515–529. Springer, 2004.

[7] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, p. 103–112, New York, NY, USA, 1988. ACM Press.

[8] D. Boneh and X. Boyen. Short signatures without random oracles. In C. Cachin and J. Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, p. 56–73. Springer, 2004.

[9] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In M. K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, p. 41–55. Springer, 2004.

[10] X. Boyen and B. Waters. Full-domain subgroup hiding and constant-size group signatures. In T. Okamoto and X. Wang, editors, *Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, p. 1–15. Springer, 2007.

[11] J. Camenisch, R. Chaabouni, and A. Shelat. Efficient protocols for set membership and range proofs. In J. Pieprzyk, editor, *ASIACRYPT*, volume 5350 of *Lecture Notes in Computer Science*, p. 234–252. Springer, 2008.

[12] J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact E-Cash. In *EUROCRYPT*, volume 3494 of LNCS, p. 302–321, 2005.

[13] J. Camenisch, G. Neven, and A. Shelat. Simulatable adaptive oblivious transfer. In M. Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, p. 573–590. Springer, 2007.

[14] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In B. Kaliski, editor, *CRYPTO '97*, volume 1296 of *LNCS*, p. 410–424. Springer Verlag, 1997.

[15] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS '01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, p. 136, Washington, DC, USA, 2001. IEEE Computer Society.

[16] R. Canetti. Obtaining universally compoable security: Towards the bare bones of trust. In *ASIACRYPT*, p. 88–112, 2007.

[17] D. Chaum. Blind signatures for untraceable payments. In *CRYPTO '82*, p. 199–203. Plenum Press, 1982.

[18] S. Coull, M. Green, and S. Hohenberger. Controlling access to an oblivious database using stateful anonymous credentials. Cryptology ePrint Archive, Report 2008/474, 2008. `http://eprint.iacr.org/`.

[19] G. D. Crescenzo, R. Ostrovsky, and S. Rajagopalan. Conditional oblivious transfer and timed-release encryption. In *EUROCRYPT*, p. 74–89, 1999.

[20] I. Damgrd, J. B. Nielsen, and C. Orlandi. Essentially optimal universally composable oblivious transfer. Cryptology ePrint Archive, Report 2008/220, 2008. `http://eprint.iacr.org/`.

[21] U. Feige, D. Lapidot, and A. Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM Journal on Computing*, 29(1):1–28, 1999.

[22] O. Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, New York, NY, USA, 2000.

[23] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[24] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[25] M. Green and S. Hohenberger. Blind identity-based encryption and simulatable oblivious transfer. In *ASIACRYPT*, p. 265–282, 2007.

[26] M. Green and S. Hohenberger. Universally composable adaptive oblivious transfer. Cryptology ePrint Archive, Report 2008/163, 2008. `http://eprint.iacr.org/`.

[27] R. Grimm and P. Aichroth. Privacy protection for signed media files: a separation-of-duty approach to the lightweight drm (lwdrm) system. In J. Dittmann and J. J. Fridrich, editors, *MM&Sec*, p. 93–99. ACM, 2004.

[28] J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In N. P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, p. 415–432. Springer, 2008.

[29] J. Herranz. Restricted adaptive oblivious transfer. Cryptology ePrint Archive, Report 2008/182, 2008. `http://eprint.iacr.org/`.

[30] Y. Ishai and E. Kushilevitz. Private simultaneous messages protocols with applications. In *In Proc. of 5th ISTCS*, p. 174–183, 1997.

[31] P. Koargonkar and L. Wolin. A multivariate analysis of web usage. *Journal of Advertising Research*, p. 53–68, March/April 1999.

[32] M. Kohlweiss, S. Faust, L. Fritsch, B. Gedrojc, and B. Preneel. Efficient oblivious augmented maps: Location-based services with a payment broker. In N. Borisov and P. Golle, editors, *Privacy Enhancing Technologies*, volume 4776 of *Lecture Notes in Computer Science*, p. 77–94. Springer, 2007.

[33] D.-G. Lee, H.-G. Oh, and I.-Y. Lee. A study on contents distribution using electronic cash system. In *EEE '04: Proceedings of the 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'04)*, p. 333–340, Washington, DC, USA, 2004. IEEE Computer Society.

[34] Y. Lindell. Lower bounds for concurrent self composition. In M. Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, p. 203–222. Springer, 2004.

[35] H. min Sun, K. hang Wang, and C. fu Hung. Towards privacy preserving digital rights management using oblivious transfer.

[36] M. Naor and B. Pinkas. Oblivious transfer with adaptive queries. In *CRYPTO*, p. 573–590, 1999.

[37] M. O. Rabin. How to exchange secrets by oblivious transfer. 1981.

[38] A. D. Santis, G. Di Crescenzo, and G. Persiano. Necessary and sufficient assumptions for non-interactive zero-knowledge proofs of knowledge for all NP relations. In U. Montanari, J. P. Rolim, and E. Welzl, editors, *Proc. 27th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1853 of *LNCS*, p. 451–462. Springer Verlag, 2000.

[39] B. Shankar, K. Srinathan, and C. P. Rangan. Alternative protocols for generalized oblivious transfer. In S. Rao, M. Chatterjee, P. Jayanti, C. S. R. Murthy, and S. K. Saha, editors, *ICDCN*, volume 4904 of *Lecture Notes in Computer Science*, p. 304–309. Springer, 2008.

[40] C. Tobias. Practical oblivious transfer protocols. In *IH '02: Revised Papers from the 5th International Workshop on Information Hiding*, p. 415–426, London, UK, 2003. Springer-Verlag.

[41] J. Tsai, S. Egelman, L. Cranor, and R. Acquisti. The effect of online privacy information on purchasing behavior: An experimental study, working paper. June 2007.

[42] D. Wagner, editor. *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*. Springer, 2008.

A Universally composable security

We review the universally composable security paradigm presented in [15]. First, we briefly explain the computational model, which is intended to represent multiple interacting computer programs. Then we review the protocol security definition: the model of protocol execution, the notion of ideal functionality and the definition of security. Finally, we recall the concept of hybrid protocol and the composition theorem.

Computational model. In order to represent a network of communicating computer programs, [15] utilizes a system of interactive Turing machines instances (ITI) that are provided with additional communication tapes which can be written into by one ITI and read by another. In contrast to an interactive Turing machine (ITM), which represents a static object (a program or algorithm), an ITI is an ITM running on some specific data. There are two methods for ITI intercommunication: communication through the communication tapes, which models untrusted communication over a network, and communication through the input and subroutine output tapes, which models local subroutine calls.

An execution of a system (I, C) is modeled as a sequence of activations of ITIs, where in each activation a single ITI is active. In the beginning, an initial ITI (I) is invoked on some external input. This ITI can perform local computations and also invoke other ITIs and write information on their corresponding tapes (in each activation, the tape of only one ITI may be written). Once an ITI is invoked, it can invoke other ITIs and write their tapes. When an ITI enters a waiting state, then the ITI whose input tape was written becomes active. If the tape of no other ITI is written, then the initial ITI is activated, and the execution ends when the initial ITI halts. There exists a control function C that determines which tapes of which ITIs can be written to by each ITI. Let $OUT_{I,C}(\kappa, x)$ denote the random variable that describes the output of the execution of the system (I, C), where κ is I's security parameter and x is I's input.

The identity of an ITI is determined at invocation time by the invoking instance and it is unchangeable. Each identity consists of two separate fields: a session id (sid) and a party id (pid). A protocol instance is a set of ITIs in a system's execution that at a certain moment have the same program and the same sid. The pid is used to differentiate ITIs within a protocol instance.

Protocol security. [15] explains the model of protocol execution in the presence of an adversary and in a given environment. This model is parameterized by three ITMs: the protocol ψ, which determines the program to be executed by parties in a protocol instance, the adversary $\mathcal{A}$, and the environment $\mathcal{Z}$, which represents all the protocols running in the system and the adversaries acting within them (including protocols that interact with ψ). $\mathcal{Z}$ is the initial ITI. The control function defines that $\mathcal{A}$ should be the first ITI to be invoked. Afterwards, $\mathcal{Z}$ can communicate with $\mathcal{A}$ or provide inputs to parties that have the program ψ and that have the same sid, which is fixed by $\mathcal{Z}$. $\mathcal{A}$ can write a message in the communication tape of a party, corrupt a party (only when instructed to do so by $\mathcal{Z}$), or send information to $\mathcal{Z}$. A party of ψ can write a message in the communication tape of $\mathcal{A}$ (and not in the one of other ITI), write outputs to the subroutine output tape of $\mathcal{Z}$, or invoke ITIs as subroutines. The execution finishes when $\mathcal{Z}$ outputs a single bit. Remarkably, we note that $\mathcal{Z}$ has access to the input and output of the parties, but neither to the communication between them nor to the inputs and outputs of their subroutines, while $\mathcal{A}$ has only access to the communication between parties. Nevertheless, $\mathcal{Z}$ and $\mathcal{A}$ can exchange information between two activations of some party.

In order to prove that a protocol ψ is secure, ψ is compared with an ideal protocol within the above model of protocol execution. The ideal protocol involves an ideal functionality $\mathcal{F}$ that acts as a trusted ITM that carries out the desired task. In the ideal protocol execution, each party forwards its input to $\mathcal{F}$ and copies any output coming from $\mathcal{F}$ to its local output. $\mathcal{F}$ has instructions to compute the desired outputs given the inputs. In addition, $\mathcal{F}$ can receive messages from adversary $\mathcal{E}$ and can be instructed to send messages to $\mathcal{E}$, which models the influence that $\mathcal{E}$ may have on the output of the parties, the information that $\mathcal{E}$ may obtain, or the delay that $\mathcal{E}$ can apply to the outputs.

A protocol ψ is secure if ψ securely realizes $\mathcal{F}$, which implies that ψ emulates the ideal protocol. Emulation means that for any $\mathcal{A}$ there exists a simulator[4] $\mathcal{E}$ such that, for any $\mathcal{Z}$ and on any input, the advantage of $\mathcal{Z}$ in guessing whether it is interacting with $\mathcal{A}$ and ψ or with $\mathcal{E}$ and the ideal protocol is negligible. More formally, let $IDEAL_{\mathcal{F},\mathcal{E},\mathcal{Z}}$ denote the ensemble of random variables $\{OUT_{\mathcal{Z},C(\mathcal{F},\mathcal{E})}(\kappa, x)\}_{\kappa\in\mathbb{N},x\in\{0,1\}^*}$ and let $REAL_{\psi,\mathcal{A},\mathcal{Z}}$ denote $\{OUT_{\mathcal{Z},C(\psi,\mathcal{A})}(\kappa, x)\}_{\kappa\in\mathbb{N},x\in\{0,1\}^*}$. ψ securely realizes $\mathcal{F}$ if these ensembles are computationally indistinguishable. Intuitively, if ψ securely realizes $\mathcal{F}$ then it is guaranteed that the outputs of the parties when running ψ are indistinguishable from their outputs when interacting with $\mathcal{F}$ on the same input, and that $\mathcal{A}$ does not learn more information when interacting with ψ than when interacting with $\mathcal{F}$.

Hybrid protocol. An $\mathcal{F}$-hybrid protocol is a protocol where parties, besides communicating with $\mathcal{A}$ as usual, make calls to instances of the ideal functionality $\mathcal{F}$ by invoking the ideal protocol of $\mathcal{F}$. $\mathcal{F}$ can be thought as an ideal service that is provided in the network.

Composition theorem. Let ψ be a protocol that makes subroutine calls to a protocol ϕ, and let ρ be a protocol that emulates ϕ. Then the composed protocol $\psi^{\rho/\phi}$, where each invocation of ϕ is replaced with an invocation of ρ, emulates ψ. As a corollary, if ψ is an $\mathcal{F}$-hybrid protocol and ρ securely realizes $\mathcal{F}$, then the composed protocol $\psi^{\rho/\mathcal{F}}$ emulates ψ. This theorem includes the traditional notion of concurrent self composition [34], where many instances of the same protocol run concurrently.

B Proof of Theorem 1

We refer to [4] for a formal definition of F-unforgeability. We consider two types of forgeries. In type 1 the forger $\mathcal{D}$ sends (s_1, s_2, s_3) and $F(m_1, \ldots, m_n)$ such that, for $q = 1$ to l, $s_1 \neq s_1^{(q)}$, where $s_1^{(q)}$ was used to answer the qth signature query from $\mathcal{D}$. In type 2, there exists q such that $s_1 = s_1^{(q)}$.

Type 1 forgeries: We construct an algorithm $\mathcal{E}$ that breaks the l-HSDH assumption with non-negligible probability if there exists a forger $\mathcal{D}$ that outputs a type 1 forgery with non-negligible probability. $\mathcal{E}$ takes as input an l-HSDH tuple that consists of $(g, g^\alpha) \in \mathbb{G}^2$, $u \in \mathbb{G}$, an l-tuple $\{g^{1/(\alpha+c_q)}, g^{c_q}, u^{c_q}\}_{q=1}^{l}$ and a description of the groups $(p, \mathbb{G}, \mathbb{G}_T)$. $\mathcal{E}$ computes a tuple $(g^{1/(\alpha+c)}, g^c, u^c)$ such that, $\forall q$, $c \neq c_q$, as follows:

$\mathsf{Setup_n}(1^k)$. $\mathcal{E}$ sets $crs_{Sig} = (p, \mathbb{G}, \mathbb{G}_T, e, g, u)$ and hands them to $\mathcal{D}$.

$\mathsf{Keygen_n}(crs_{Sig})$. $\mathcal{E}$ picks random $(\beta_1, \ldots, \beta_n, \lambda_1, \ldots, \lambda_n) \leftarrow \mathbb{Z}_p$ and calculates a public key $Pk = (v, g_1, \ldots, g_n, u_1, \ldots, u_n) = (g^\alpha, g^{\beta_1}, \ldots, g^{\beta_n}, u^{\lambda_1}, \ldots, u^{\lambda_n})$. The secret key is $(\alpha, \beta_1, \ldots, \beta_n) \leftarrow \mathbb{Z}_p$, although $\mathcal{D}$ does not know α. $\mathcal{E}$ sends Pk to $\mathcal{D}$.

$\mathcal{O}\mathsf{Sign_n}(crs_{Sig}, Sk, \vec{m})$. At the qth query, $q \in \{1, \ldots, l\}$, $\mathcal{E}$ implicitly sets $c_q = r + \sum_{i=1}^{n} \beta_i m_i$ and computes (s_1, s_2, s_3) as follows:

$$s_1 = g^{1/(\alpha+c_q)}$$
$$s_2 = g^{c_q}/g^{\sum_{i=1}^{n}\beta_i m_i} = g^r$$
$$s_3 = u^{c_q}/u^{\sum_{i=1}^{n}\beta_i m_i} = u^r$$

$\mathcal{E}$ sends (s_1, s_2, s_3) to $\mathcal{D}$.

[4] Adversary $\mathcal{E}$ is often called simulator because typically in security proofs $\mathcal{E}$ operates by simulating $\mathcal{A}$.

Forgery. $\mathcal{D}$ outputs a forgery $(s_1, s_2, s_3) = (g^{1/(\alpha+r+\beta_1 m_1+\ldots+\beta_n m_n)}, g^r, u^r)$ and $(g_1^{m_1}, u_1^{m_1}, \ldots, g_n^{m_n}, u_n^{m_n})$. $\mathcal{E}$ implicitly sets $c = r + \sum_{i=1}^{n} \beta_i m_i$ and computes an HSDH tuple (A, B, C) as follows:

$$A = s_1 = g^{1/(\alpha+c)}$$
$$B = s_2 \prod_{i=1}^{n} g_i^{m_i} = g^r \prod_{i=1}^{n} g^{m_i \beta_i} = g^c$$
$$C = s_3 \prod_{i=1}^{n} (u_i^{m_i})^{(\beta_i/\lambda_i)} = u^r \prod_{i=1}^{n} (u^{\lambda_i m_i})^{(\beta_i/\lambda_i)} = u^c$$

Type 2 forgeries: We construct an algorithm $\mathcal{E}$ that breaks the l-TDH assumption with non-negligible probability if there exists a forger $\mathcal{D}$ that outputs a type 2 forgery with non-negligible probability. $\mathcal{E}$ takes as input a l-TDH tuple that consists of $(g, g^x, g^y) \in \mathbb{G}^3$, an l-tuple $\{c_q, g^{1/(x+c_q)}\}_{q=1}^{l}$ and a description of the groups $(p, \mathbb{G}, \mathbb{G}_T)$. $\mathcal{E}$ computes a $(g^{\mu x}, g^{\mu y}, g^{\mu x y})$ as follows:

$\mathsf{Setup_n}(1^k)$. $\mathcal{E}$ sets $crs_{Sig} = (p, \mathbb{G}, \mathbb{G}_T, e, g, u)$, where $u = g^y$, and hands them to $\mathcal{D}$.

$\mathsf{Keygen_n}(crs_{Sig})$. $\mathcal{E}$ picks random $t \leftarrow \{1, \ldots, n\}$, $(\alpha, \{\beta_i\}_{i=1, i\neq t}^{n}, \{\lambda_i\}_{i=1}^{n}) \leftarrow \mathbb{Z}_p$ and computes $Pk = (v, g_1, \ldots, g_n, u_1, \ldots, u_n) = (g^\alpha, g^{\beta_1}, \ldots, g^{\beta_n}, u^{\lambda_1}, \ldots, u^{\lambda_n})$ by setting $g_t = g^{x\gamma}$ for random $\gamma \leftarrow \mathbb{Z}_p$. The secret key is $(\alpha, \beta_1, \ldots, \beta_n)$, although $\mathcal{E}$ does not know $\beta_t = x\gamma$. $\mathcal{E}$ sends Pk to $\mathcal{D}$.

$\mathcal{O}\mathsf{Sign_n}(crs_{Sig}, Sk, \vec{m})$. In the qth query, $\mathcal{E}$ sets $c_q = (\alpha + r + \sum_{i=1, i\neq t}^{n} \beta_i m_i)/\gamma m_t$, where $q \in \{1, \ldots, l\}$. Then $(x + c_q)\gamma m_t = \alpha + r + \sum_{i=1}^{n} \beta_i m_i$ is the inverse of the exponent that should be used to compute s_1. Therefore, $\mathcal{E}$ sets $r = c_q \gamma m_t - \alpha - \sum_{i=1, i\neq t}^{n} \beta_i m_i$ and computes $(s_1, s_2, s_3) = ((g^{1/(x+c_q)})^{(1/\gamma m_t)}, g^r, u^r)$. $\mathcal{E}$ sends (s_1, s_2, s_3) to $\mathcal{D}$.

Forgery. $\mathcal{D}$ outputs a forgery $(s_1, s_2, s_3) = (g^{1/(\alpha+r+\beta_1 m_1+\ldots+\beta_n m_n)}, g^r, u^r)$ and $(g_1^{m_1}, u_1^{m_1}, \ldots, g_n^{m_n}, u_n^{m_n})$. $\mathcal{E}$ already has a message-signature pair such that $r + \sum_{i=1}^{n} \beta_i m_i = r^{(q)} + \sum_{i=1}^{n} \beta_i m_i^{(q)}$, but there exists i such that $m_i \neq m_i^{(q)}$. If $i = t$, then $m_t \neq m_t^{(q)}$ but $\beta_t m_t + r + \sum_{i=1, i\neq t}^{n} \beta_i m_i = \beta_t m_t^{(q)} + r^{(q)} + \sum_{i=1, i\neq t}^{n} \beta_i m_i^{(q)}$. $\mathcal{E}$ implicitly sets $\mu = (m_t^{(q)} - m_t)\gamma$. $\mathcal{E}$ computes a TDH tuple (A, B, C) as follows:

$$A = (u^{m_t{}^{(q)}}/(u_t^{m_t})^{1/\lambda_t})^\gamma = (u^{m_t^{(q)} - m_t})^\gamma = u^\mu = g^{y\mu}$$
$$B = \prod_{i=1, i\neq t}^{m} (g_i^{m_i}/g_i^{m_i^{(q)}})(s_2/g^{r^{(q)}}) = g^{\sum_{i=1, i\neq t}^{n} \beta_i(m_i - m_i^{(q)})} g^{(r - r^{(q)})} =$$
$$g^{\beta_t(m_t^{(q)} - m_t)} = g^{x\gamma(m_t^{(q)} - m_t)} = g^{x\mu}$$
$$C = \prod_{i=1, i\neq t}^{m} ((u_i^{m_i})^{1/\lambda_i}/u^{m_i^{(q)}})^{\beta_i}(s_3/u^{r^{(q)}}) = u^{\sum_{i=1, i\neq t}^{n} \beta_i(m_i - m_i^{(q)})} u^{(r - r^{(q)})} =$$
$$u^{\beta_t(m_t^{(q)} - m_t)} = u^{x\gamma(m_t^{(q)} - m_t)} = u^{x\mu} = g^{xy\mu}$$

C Proof of Theorem 2

In order to prove this theorem, we need to build a simulator $\mathcal{E}$ that invokes a copy of adversary $\mathcal{A}$ and interacts with $\mathcal{F}_{POT}$ and environment $\mathcal{Z}$ in such a way that ensembles $IDEAL_{\mathcal{F}_{POT}, \mathcal{E}, \mathcal{Z}}$ and $REAL_{POT, \mathcal{A}, \mathcal{Z}}$ are computationally indistinguishable.

In our proof we make use of the fact that Groth-Sahai proofs are partially extractable, composable witness-indistinguishable, and (given certain conditions) composable zero-knowledge. The following algorithms formalize these properties and are needed for the security proofs.

$\mathsf{PKExtractSetup}(1^\kappa)$**.** It outputs a tuple crs_{PK} that is identically distributed to the output of $\mathsf{PKSetup}(1^\kappa)$ and an extraction trapdoor td_{ext}.

$\mathsf{PKExtract}(crs_{PK}, td_{ext}, y, pok)$**.** It uses td_{ext} to extract the witnesses w from pok. The algorithm does not extract the openings of the commitments.

$\mathsf{PKSimSetup}(1^\kappa)$**.** It outputs an alternative setup crs'_{PK} and a simulation trapdoor td_{sim}.

$\mathsf{PKSimProve}(crs'_{PK}, td_{sim}, y)$**.** Given simulation parameters crs'_{PK}, it outputs a proof pok for instance y such that $\mathsf{PKVerify}(crs'_{PK}, y, pok)$ outputs accept.[5]

Groth-Sahai proofs fulfill the correctness property. $(\mathsf{PKExtSetup}, \mathsf{PKExtract})$ are a polynomial time extractor that allows for perfect extractability of the witness, i.e., extraction is done with probability 1. However, if we have a commitment $\mathsf{Commit}(x, open)$, then $\mathsf{PKExtract}$ extracts x but not the opening $open$.

Groth-Sahai proofs are composable witness-indistinguishable, but only for some classes of statements they fulfill the stronger notion of composable zero-knowledge.

Composable Witness-Indistinguishability. We require two properties: (1) the parameters crs'_{PK} output by $\mathsf{PKSimSetup}$ are computationally indistinguishable from crs_{PK} output by $\mathsf{PKSetup}$, and (2) all (information theoretic) adversaries $\mathcal{A}$ have advantage 0 in the following game:

1. $\mathcal{A}$ receives crs'_{PK} as generated by $\mathsf{PKSimSetup}$.
2. $\mathcal{A}$ outputs an instance y and two valid witnesses (w_0, w_1).
3. $\mathcal{A}$ receives a proof $pok = \mathsf{PKProve}(crs'_{PK}, y, w_b)$, where b is a random bit.
4. $\mathcal{A}$ sends its guess b'. Its advantage is $|Pr[b = b'] - 1/2|$.

Composable Zero-Knowledge. There exists a simulator $(\mathsf{PKSimSetup}, \mathsf{PKSimProve})$ that fulfills two properties: (1) the parameters crs'_{PK} output by $\mathsf{PKSimSetup}$ are computationally indistinguishable from the parameters crs_{PK} output by $\mathsf{PKSetup}$, and (2) all (information theoretic) adversaries $\mathcal{A}$ have advantage 0 in the following game:

1. $\mathcal{A}$ receives crs'_{PK} and td_{sim} as generated by $\mathsf{PKSimSetup}$.
2. $\mathcal{A}$ outputs an instance y and a valid witness w.
3. Let $pok_0 = \mathsf{PKProve}(crs'_{PK}, y, w)$ and $pok_1 = \mathsf{PKSimProve}(crs'_{PK}, td_{sim}, y)$. Pick a random bit b. $\mathcal{A}$ receives pok_b.
4. $\mathcal{A}$ sends its guess b'. Its advantage is $|Pr[b = b'] - 1/2|$.

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

1. $\mathcal{E}$ runs algorithm $\mathsf{PKSetup}$ to generate two Groth-Sahai reference strings $crs^{\mathcal{B}}_{PK}$ and $crs^{\mathcal{V}}_{PK}$ for the same pairing group setup $(p, \mathbb{G}, \mathbb{G}_T, e, g)$, where $-p_{max} > A \mod p$ holds. $\mathcal{E}$ picks random $u \leftarrow \mathbb{G}$. $\mathcal{E}$ picks random $a, b, c \leftarrow \mathbb{Z}_p$, first computes $h_3 = g^c$ and then computes $(h_1, h_2) = (h_3^{1/a}, h_3^{1/b})$. $\mathcal{E}$ sets $crs = (crs^{\mathcal{V}}_{PK}, crs^{\mathcal{B}}_{PK}, u, h_1, h_2, h_3)$. When $\mathcal{F}_{CRS}$ is queried, $\mathcal{E}$ returns (sid, crs).

[5] The statement y should belong to the class of statements for which the Groth-Sahai proof system can compute a zero-knowledge proof.

2. Upon receiving (sid, T) from $\mathcal{A}$, $\mathcal{E}$ verifies T as explained in algorithm POTInitBuyer and aborts when not all the checks verify. Otherwise $\mathcal{E}$ parses T as $(pk, C_1, \ldots, C_N)$. For $i = 1, \ldots, N$, $\mathcal{E}$ parses C_i to get (c_3, c_4, c_5, p_i) and sets $m_i = c_5/(c_3^a c_4^b)$. $\mathcal{E}$ sends $(sid, vendor, m_1, \ldots, m_N, p_1, \ldots, p_N)$ to $\mathcal{F}_{POT}$.

3. Upon receiving (sid, ac_0) from $\mathcal{F}_{POT}$, $\mathcal{E}$ computes $(P, D_0^{(priv)})$ as explained in POTInitBuyer. $\mathcal{E}$ sends (sid, P) to $\mathcal{A}$ and keeps $D_0^{(priv)}$.

4. In the ith transfer phase, upon receiving $(sid, request)$ from $\mathcal{F}_{POT}$, $\mathcal{E}$ executes POTRequest$(crs, T, D_{i-1}^{(priv)}, \sigma_{min})$, where σ_{min} corresponds to the message with the lowest price, to obtain $(Q, Q^{(priv)}, D_i^{(priv)})$, and sends Q to $\mathcal{A}$. Upon receiving the response R from $\mathcal{A}$, $\mathcal{E}$ runs POTComplete $(crs, T, R, Q^{(priv)})$. If the output is reject, $\mathcal{E}$ sends $(sid, 0)$ to $\mathcal{F}_{POT}$. Otherwise, $\mathcal{E}$ sends $(sid, 1)$ to $\mathcal{F}_{POT}$ and keeps $D_i^{(priv)}$.

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

1. $\mathcal{E}$ runs algorithm PKExtractSetup to obtain $crs_{PK}^{\mathcal{B}}$ and an extraction trapdoor td_{ext}, and algorithm PKSimSetup to obtain $crs_{PK}^{\mathcal{V}}$ and a simulation trapdoor td_{sim}. Both use the same pairing group setup $(p, \mathbb{G}, \mathbb{G}_T, e, g)$, where $-p_{max} > A \mod p$ holds. $\mathcal{E}$ picks random $a, b, c \leftarrow \mathbb{Z}_p$ and computes $(h_1, h_2, h_3) = (g^a, g^b, g^c)$. $\mathcal{E}$ picks random $u \leftarrow \mathbb{G}$ and sets $crs = (crs_{PK}^{\mathcal{V}}, crs_{PK}^{\mathcal{B}}, u, h_1, h_2, h_3)$. $\mathcal{E}$ returns (sid, crs) when $\mathcal{F}_{CRS}$ is queried.

2. Upon receiving $(sid, p_1, \ldots, p_N)$ from $\mathcal{F}_{POT}$, $\mathcal{E}$ picks random messages $m'_1, \ldots, m'_N \in \mathbb{G}^N$ and runs POTInitVendor$(crs, m'_1, \ldots, m'_N, p_1, \ldots, p_N, A)$ to obtain T. $\mathcal{E}$ sends (sid, T) to $\mathcal{A}$.

3. Upon receiving (sid, P), $\mathcal{E}$ runs $(D_0, ac_0) \leftarrow$ POTGetDeposit(crs, P, A), sends $(sid, buyer, ac_0)$ to $\mathcal{F}_{POT}$ and keeps D_0.

4. In the ith transfer phase, upon receiving (sid, Q) from $\mathcal{A}$, $\mathcal{E}$ parses Q as (d_1, d_2, pok_1, D_i). $\mathcal{E}$ verifies the proof pok_1 by running PKVerify and utilizing (D_i, D_{i-1}), and aborts if verification fails. Otherwise, $\mathcal{E}$ executes PKExtract$(crs_{PK}^{\mathcal{B}}, td_{ext}, pok_1)$ to extract the witness. Then, for $i = 1$ to N, $\mathcal{E}$ compares the signature $(g_1^{r_1}, u_1^{r_1}, g_2^{r_2}, u_2^{r_2}, g_3^{p_i}, u_3^{p_i}, s_1, s_2, s_3)$ in the witness with each of the signatures that are included in the ciphertexts that were sent to $\mathcal{A}$ in order to know the choice σ_i selected by $\mathcal{A}$. $\mathcal{E}$ also compares the signatures $\{g^{\sigma_j}, u^{\sigma_j}, S_{\sigma_j}\}_{j=0}^{a-1}$ in the witness that correspond to the range proof with each of the signatures that were sent to $\mathcal{A}$ in $params_{Range}$. (This is done in order to ensure that $\mathcal{A}$ did not compute a forgery.) $\mathcal{E}$ stores D_i and sends $(sid, buyer, \sigma_i)$ to $\mathcal{F}_{POT}$ in order to obtain either $\perp$ or the message m_{σ_i}. For the former, $\mathcal{E}$ sends $(sid, \perp)$ to $\mathcal{A}$. Otherwise $\mathcal{E}$ 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 $z = (c_5 t_1 t_2)/m_{\sigma_i}$, and uses trapdoor td_{sim} to simulate proof pok_2. $\mathcal{E}$ sets $R = (z, pok_2)$ and sends (sid, R) to $\mathcal{A}$.

Simulation when none of the parties is corrupted. After receiving $(sid, p_1, \ldots, p_N)$ and k messages of the form (sid, b), $\mathcal{E}$ creates a simulated transcript by running copies of honest $\mathcal{V}$ and $\mathcal{B}$. $\mathcal{V}$ is run on input random messages $(m'_1, \ldots, m'_N)$ and prices $(p_1, \ldots, p_N)$ while $\mathcal{B}$ is run on input an account ac_0 such that $ac_0 > p_{\sigma_{min}} k$, where σ_{min} denotes the item with the lowest price. In the ith transfer phase, $\mathcal{B}$ receives as input σ_{min}. If $b_i = 0$ then $\mathcal{V}$ sends a invalid response $(sid, \perp)$. Otherwise $\mathcal{V}$ sends a valid response.

Simulation when $\mathcal{V}$ and $\mathcal{B}$ are corrupted. In this case $\mathcal{E}$ knows the inputs to $\mathcal{B}$ and $\mathcal{V}$ and so $\mathcal{E}$ can simulate by computing the real messages that are sent by the two parties.

Claim 1 (Buyer security) *When only $\mathcal{V}$ is corrupted, then $IDEAL_{\mathcal{F}_{POT},\mathcal{E},\mathcal{Z}}$ and $REAL_{POT,\mathcal{A},\mathcal{Z}}$ are computationally indistinguishable under the DLIN assumption.*

Proof. We show by means of a series of hybrid games that the environment $\mathcal{Z}$ cannot distinguish between the real execution ensemble $REAL_{POT,\mathcal{A},\mathcal{Z}}$ and the simulated ensemble $IDEAL_{\mathcal{F}_{POT},\mathcal{E},\mathcal{Z}}$ with non-negligible probability. We denote by Pr [**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 [**Game** 0] $= 0$.

Game 1**:** This game proceeds as **Game** 0, except that to generate crs we pick random $a, b, c \leftarrow \mathbb{Z}_p$, first compute $h_3 = g^c$ and then compute $(h_1, h_2) = (h_3^{1/a}, h_3^{1/b})$. crs is set to $(crs^{\mathcal{V}}_{PK}, crs^{\mathcal{B}}_{PK}, u, h_1, h_2, h_3)$. Since this crs has the same distribution as in **Game** 0, then |Pr [**Game** 1] $-$ Pr [**Game** 0]| $=$ 0.

Game 2**:** This game proceeds as **Game** 1, except that message $P = (w_1^{r_1}, w_2^{r_2}, ac_0 \cdot h_3^{r_1+r_2})$ is replaced by another valid message that is computed by using the same value ac_0, so both messages are identically distributed. Therefore, |Pr [**Game** 2] $-$ Pr [**Game** 1]| $= 0$.

Game 3**:** This game differs from the previous one in that in each transfer phase the request $Q = (d_1, d_2, pok_1, D_i)$ is computed by running $\mathsf{POTRequest}(crs, T, D_{i-1}^{(priv)}, \sigma_{min})$, where σ_{min} is the message with the lowest price. Since the values (d_1, d_2) are uniformly distributed over $\mathbb{Z}_p$ and (pok_1, D_i) are computationally witness indistinguishable under the DLIN assumption, then Q cannot be distinguished from a request computed by using another selection value $\sigma \in \{1, \dots, N\}$. Therefore, |Pr [**Game** 3] $-$ Pr [**Game** 2]| $\leq \nu(\kappa)$.

$\mathcal{E}$ performs all the changes described in **Game** 3, but, for $i = 1$ to N, $\mathcal{E}$ uses the ciphertexts that are sent by $\mathcal{A}$ to compute messages $m_i = c_5/(c_3^a c_4^b)$, gets p_i and sends $(sid, vendor, m_1, \dots, m_N, p_1, \dots, p_N)$ to $\mathcal{F}_{POT}$. Upon receiving ac_0 from $\mathcal{F}_{POT}$, $\mathcal{E}$ computes P by following $\mathsf{POTInitBuyer}$, and stores private state $D_0^{(priv)}$. In the ith transfer, $\mathcal{E}$ computes a request for σ_{min} by using $D_{i-1}^{(priv)}$ and stores private information $D_i^{(priv)}$. $\mathcal{E}$ also plays the role of the verifier when $\mathcal{A}$ sends the response. (We note that, since we use the item with the lowest price, $\mathcal{A}$ never rejects because there are not enough funds. Note that $\mathcal{F}_{POT}$ asks the vendor whether he wants to make the transfer fail after checking that the buyer has enough funds.) If the response is not valid, $\mathcal{E}$ sends $b = 0$ to $\mathcal{F}_{POT}$. Otherwise $\mathcal{E}$ sends $b = 1$ to $\mathcal{F}_{POT}$. The distribution produced in **Game** 3 is identical to that of our simulation. Therefore, we have that |Pr [**Game** 3] $\leq \nu(\kappa)$.

Claim 2 (Vendor security) *When only $\mathcal{B}$ is corrupted, then $IDEAL_{\mathcal{F}_{OT},\mathcal{E},\mathcal{Z}}$ and $REAL_{OT,\mathcal{A},\mathcal{Z}}$ are computationally indistinguishable under the* $(\max(N, d))$*-HSDH,* $(\max(N, d))$*-TDH and DLIN assumptions.*

Proof. We show by means of a series of hybrid games that the environment $\mathcal{Z}$ cannot distinguish between the real execution ensemble $REAL_{POT,\mathcal{A},\mathcal{Z}}$ and the simulated ensemble $IDEAL_{\mathcal{F}_{POT},\mathcal{E},\mathcal{Z}}$ with non-negligible probability. We again denote by Pr [**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 $\mathcal{V}$. Therefore, Pr [**Game** 0] $= 0$.

Game 1**:** This game proceeds as **Game** 0, except that to set crs we run PKExtractSetup to obtain $crs^{\mathcal{B}}_{PK}$ and an extraction trapdoor td_{ext}, and PKSimSetup to obtain $crs^{\mathcal{V}}_{PK}$ and a simulation trapdoor td_{sim}. Both use the same pairing group setup $(p, \mathbb{G}, \mathbb{G}_T, e, g)$, where $-p_{max} > A \mod p$ holds. We pick random $a, b, c \leftarrow \mathbb{Z}_p$ and compute $(h_1, h_2, h_3) = (g^a, g^b, g^c)$. We also pick random $u \leftarrow \mathbb{G}$ and set $crs = (crs^{\mathcal{V}}_{PK}, crs^{\mathcal{B}}_{PK}, u, h_1, h_2, h_3)$. $crs^{\mathcal{B}}_{PK}$ computed as above is identically distributed to the output of PKSetup. If the DLIN assumption holds, then $crs^{\mathcal{V}}_{PK}$ generated as above is computationally indistinguishable from that generated by PKSetup, and thus |Pr [**Game** 1] $-$ Pr [**Game** 0]| $\leq \nu_1(\kappa)$.

Game 2**:** The difference between this game and the previous one consists in that in each transfer phase we extract the witness of pok_1 by running PKExtract$(crs_{PK}, td_{ext}, y, pok_1)$. Because Groth-Sahai proofs are perfectly extractable, i.e., extraction never fails, we have that |Pr [**Game** 2] $-$ Pr [**Game** 1]| $= 0$.

Game 3**:** This game is identical to **Game** 2, except that **Game** 3 aborts if the extracted values that correspond to the signature $(g_1^{r_1}, u_1^{r_1}, g_2^{r_2}, u_2^{r_2}, g_3^{p_i}, u_3^{p_i}, s_1, s_2, s_3)$ do not equal to any of the signatures included in the ciphertexts $C_1, \ldots, C_N$ that where sent to $\mathcal{A}$ in the initialization phase. This means that $\mathcal{A}$ computed a forged signature of the multi-block P-Signature scheme. Otherwise, we get the selection value σ_i. The probability that $\mathcal{Z}$ distinguishes between **Game** 2 and **Game** 3 is bounded by the following lemma:

Lemma 1 *If the N-HSDH and the N-TDH assumptions hold, then |Pr [**Game** 3]$-$Pr [**Game** 2]| $=$ $\nu_2(\kappa)$.*

Proof. We construct a forger $\mathcal{D}$ that breaks the F-unforgeability of the multi-block P-signature scheme with non-negligible probability. Given such a forger, in Proof of Theorem 1 we show how to construct an algorithm $\mathcal{E}$ that breaks either the N-HSDH assumption or the N-TDH assumption with non-negligible probability.

Given a buyer $\mathcal{B}$ that causes **Game** 3 to abort with non-negligible probability, $\mathcal{D}$ works as follows:

1. $\mathcal{D}$ obtains the parameters of the signature scheme $crs_{Sig} = (p, \mathbb{G}, \mathbb{G}_T, e, g, u)$ from $\mathcal{E}$, runs PKExtractSetup to get $(crs^{\mathcal{B}}_{PK}, td_{ext})$, PKSetup to get $crs^{\mathcal{V}}_{PK}$, and sets $(h_1, h_2, h_3) = (g^a, g^b, g^c)$. $\mathcal{D}$ sets $crs = (crs^{\mathcal{V}}_{PK}, crs^{\mathcal{B}}_{PK}, u, h_1, h_2, h_3)$.
2. $\mathcal{D}$ obtains the public key of the signature scheme Pk from $\mathcal{E}$.
3. For $i = 1$ to N, $\mathcal{D}$ picks random $r_1, r_2 \leftarrow \mathbb{Z}_p$ and queries $\mathcal{E}$ to obtain a signature (s_1, s_2, s_3) on the message $m_i = (r_1, r_2, p_i)$. Then $\mathcal{D}$ follows algorithm POTInitVendor to compute $T = (pk, C_1, \ldots, C_N)$.
4. Upon receiving a request $Q = (d_1, d_2, pok_1, D_i)$ from $\mathcal{B}$, $\mathcal{D}$ extracts the witness pok_1. If the extracted values that correspond to the signature $(g_1^{r_1}, u_1^{r_1}, g_2^{r_2}, u_2^{r_2}, g_3^{p_i}, u_3^{p_i}, s_1, s_2, s_3)$ do not equal to any of the signatures included in the ciphertexts $(C_1, \ldots, C_N)$, $\mathcal{D}$ outputs $(g_1^{r_1}, u_1^{r_1}, g_2^{r_2}, u_2^{r_2}, g_3^{p_i}, u_3^{p_i}, s_1, s_2, s_3)$ as a forgery.

Game 4**:** This game is identical to **Game** 3, except that **Game** 4 aborts if at least one of the extracted signatures that is employed in the range proof $\{g^{\sigma_j}, u^{\sigma_j}, S_{\sigma_j}\}_{j=0}^{a-1}$ does not equal any of the signatures $\{(g^i, u^i, S_i)\}_{i \in \mathbb{Z}_d}$, where the signatures S_i are included in $params_{Range}$. This means that $\mathcal{A}$ computed a forged signature of the single-message P-Signature scheme. The probability that $\mathcal{Z}$ distinguishes between **Game** 3 and **Game** 4 is bounded by the following lemma:

Lemma 2 *If the d-HSDH and the d-TDH assumptions hold, then* $|Pr\ [\textbf{\textit{Game}}\ 4] - Pr\ [\textbf{\textit{Game}}\ 3]| = \nu_3(\kappa)$.

Proof. We build a forger $\mathcal{D}$ that breaks the F-unforgeability of the single-message P-signature scheme with non-negligible probability. Given such a forger, in [4] it is shown how to construct an algorithm $\mathcal{E}$ that breaks either the d-HSDH assumption or the d-TDH assumption with non-negligible probability.

Given a buyer that causes **Game** 4 to abort with non-negligible probability, $\mathcal{D}$ works as follows:

1. $\mathcal{D}$ obtains the parameters of the signature scheme $crs_{Sig} = (p, \mathbb{G}, \mathbb{G}_T, e, g, u)$ from $\mathcal{E}$, runs $\mathsf{PKExtractSetup}$ to get $(crs_{PK}^{\mathcal{B}}, td_{ext})$, $\mathsf{PKSetup}$ to get $crs_{PK}^{\mathcal{V}}$, and computes $(h_1, h_2, h_3) = (g^a, g^b, g^c)$. $\mathcal{D}$ sets $crs = (crs_{PK}^{\mathcal{V}}, crs_{PK}^{\mathcal{B}}, u, h_1, h_2, h_3)$.
2. $\mathcal{D}$ obtains the public key of the signature scheme Pk from $\mathcal{E}$.
3. For $i \in \mathbb{Z}_d$, $\mathcal{D}$ queries $\mathcal{E}$ to obtain a signature S_i on the message i. $\mathcal{D}$ uses these signatures to set $params_{Range}$. Then $\mathcal{D}$ follows algorithm $\mathsf{POTInitVendor}$ to compute $T = (pk, C_1, \ldots, C_N)$.
4. Upon receiving a request $Q = (d_1, d_2, pok_1, D_i)$ from $\mathcal{B}$, $\mathcal{D}$ extracts the witnesses that correspond to the signatures $\{g^{\sigma_j}, u^{\sigma_j}, S_{\sigma_j}\}_{j=0}^{a-1}$ that are employed in the range proof. If there exists a signature $\{g^{\sigma_j}, u^{\sigma_j}, S_{\sigma_j}\}$ that does not equal any of the signatures $\{(g^i, u^i, S_i)\}_{i \in \mathbb{Z}_d}$ in $params_{Range}$, then $\mathcal{D}$ outputs this tuple $\{g^{\sigma_j}, u^{\sigma_j}, S_{\sigma_j}\}$ as a forgery.

Game 5**:** The response R is computed as (z', pok_2'), where 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 are used to compute a response $z' = (c_5 t_1 t_2)/m_{\sigma_i}$. We can see that $z' = h_3^{r_1+y_1} h_3^{r_2+y_2} = (w_1^{r_1+y_1})^{x_1}(w_2^{r_2+y_2})^{x_2} = d_1^{x_1} d_2^{x_2} = z$, where z is the honestly generated response. The proof pok_2' is computed by running $\mathsf{PKSimProve}$ and using td_{sim}. A simulated proof is indistinguishable from a proof computed by algorithm $\mathsf{PKProve}$ under the DLIN assumption. Therefore, the probability that $\mathcal{Z}$ distinguishes between **Game** 5 and **Game** 4 is bounded by $|\text{Pr}\ [\textbf{Game}\ 5] - \text{Pr}\ [\textbf{Game}\ 4]| = \nu_4(\kappa)$.

Game 6**:** In this game the messages $(m_1, \ldots, m_N)$ are replaced by random elements $(m_1', \ldots, m_N')$ of $\mathbb{G}$ when computing $\mathsf{POTInitVendor}$. Now pok_2' is a proof of an invalid statement, and is simulated by using td_{sim}. The probability that $\mathcal{Z}$ distinguishes between **Game** 6 and **Game** 5 is bounded by the following claim:

Lemma 3 *If the DLIN assumption holds,* $|Pr\ [\textbf{\textit{Game}}\ 6] - Pr\ [\textbf{\textit{Game}}\ 5]| = \nu_5(\kappa)$.

Proof. We construct an algorithm $\mathcal{T}$ that breaks the DLIN assumption given an environment $\mathcal{Z}$ that distinguishes **Game** 5 from **Game** 6 with non-negligible probability. On input a DLIN tuple $(g, g^a, g^b, g^{ac}, g^{bd}, z)$, $\mathcal{T}$ works as follows:

1. $\mathcal{T}$ sets $(g_1, g_2) = (g^a, g^b)$. $\mathcal{T}$ picks random $\delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_6 \leftarrow \mathbb{Z}_p$ and computes $(w_1, w_2) = (g_1^{\delta_1}, g_2^{\delta_2})$, $(u_1, u_2) = (g_1^{\delta_3}, g_2^{\delta_4})$ and $(h_1, h_2) = (g_1^{\delta_5}, g_2^{\delta_6})$.
2. $\mathcal{T}$ picks random $\gamma, \mu \leftarrow \mathbb{Z}_p$ and computes $h_3 = g^{\gamma}$ and $u = g^{\mu}$. $\mathcal{T}$ runs $\mathsf{PKExtractSetup}$ and $\mathsf{PKSimSetup}$ to obtain $(crs_{PK}^{\mathcal{B}}, td_{ext})$ and $(crs_{PK}^{\mathcal{V}}, td_{sim})$ respectively. $\mathcal{T}$ sets $crs = (crs_{PK}^{\mathcal{V}}, crs_{PK}^{\mathcal{B}}, u, h_1, h_2, h_3)$.
3. $\mathcal{T}$ picks random $(\alpha, \beta_3, \lambda_3) \leftarrow \mathbb{Z}_p$ and computes $(v, g_3, u_3) = (g^{\alpha}, g^{\beta_3}, u^{\lambda_3})$. $\mathcal{T}$ sets $Pk = (v, g_1, g_2, g_3, u_1, u_2, u_3)$.

4. $\mathcal{T}$ computes $params_{Range}$ as usual and sets $pk = (w_1, w_2, Pk, params_{Range})$.
5. For $i = 1$ to N, $\mathcal{T}$ picks random $\upsilon_i, \tau_{1i}, \tau_{2i} \leftarrow \mathbb{Z}_p$ and sets $(c_6, c_7) = (g^{ac\upsilon_i} g^{a\upsilon_i\tau_{1i}}, g^{bd\upsilon_i} g^{b\upsilon_i\tau_{2i}})$. This implicitly sets $(r_1, r_2) = (\upsilon_i(c + \tau_{1i}), \upsilon_i(d + \tau_{2i}))$. $\mathcal{T}$ picks random $\phi_i \leftarrow \mathbb{Z}_p$ and computes $(s_1, s_2, s_3) = (g^{1/(\alpha+\phi_i)}, g^{\phi_i}/(c_6 c_7 g_3^{p_i}), u^{\phi_i}/(c_6 c_7 g_3^{p_i})^{\mu})$. $\mathcal{T}$ sets $C_i = (c_6^{\delta_1}, c_7^{\delta_2}, c_6^{\delta_5}, c_7^{\delta_6}, z^{\gamma\upsilon_i} h_3^{\upsilon_i(\tau_{1i}+\tau_{2i})} m_i, c_6, c_7, c_6^{\delta_3}, c_7^{\delta_4}, s_1, s_2, s_3)$.
6. $\mathcal{T}$ sets $T = (pk, C_1, \ldots, C_N)$, sends it and thereafter answers requests.

As can be seen, if $z = g^{c+d}$, then, for $i = 1$ to N, C_i perfectly encrypts message m_i. However, if z is a random element of $\mathbb{G}$, then $\mathcal{T}$ encrypts random messages. Consequently, if there exists an environment $\mathcal{Z}$ that distinguishes between these two cases with non-negligible probability, then $\mathcal{T}$ breaks the DLIN assumption with non-negligible probability $\nu_5(\kappa)$.

$\mathcal{E}$ performs all the changes described in **Game** 6, but in the initialization phase $\mathcal{E}$ runs $\mathsf{POTGetDeposit}$ and sends ac_0 to $\mathcal{F}_{POT}$, and in each transfer phase $\mathcal{E}$ sends the extracted value σ_i to $\mathcal{F}_{POT}$ to obtain either $\perp$ or message m_{σ_i}. In the latter case, $\mathcal{E}$ uses the message m_{σ_i} to compute R. The distribution produced in **Game** 6 is identical to that of our simulation. Therefore, by summation we have that |Pr [**Game** 6] $\leq \nu_6(\kappa)$.

Claim 3 (Security when $\mathcal{V}$ and $\mathcal{B}$ are corrupted) *When $\mathcal{V}$ and $\mathcal{B}$ are corrupted, then $IDEAL_{\mathcal{F}_{OT},\mathcal{E},\mathcal{Z}}$ and $REAL_{OT,\mathcal{A},\mathcal{Z}}$ are indistinguishable.*

In this case $\mathcal{E}$ knows the inputs of both parties and so $\mathcal{E}$ can compute the real messages exchanged between the two parties.

Claim 4 (Security when none of the parties is corrupted) *When none of the parties is corrupted, then $IDEAL_{\mathcal{F}_{OT},\mathcal{E},\mathcal{Z}}$ and $REAL_{OT,\mathcal{A},\mathcal{Z}}$ are computationally indistinguishable under the DLIN assumption.*

We do not provide a formal proof of this claim. In the initialization phase, the $\mathcal{V}$'s message consists of a random database, which we demonstrated that is indistinguishable from a real database under the DLIN assumption. The $\mathcal{B}$'s message consists of the encryption of a different account value. We note that this encryption is of the same form as the one used to compute the ciphertexts of the database, and so it is secure under the DLIN assumption. In the transfer phase, the request message is replaced by a valid request message for message σ_{min}. Under the DLIN assumption, Groth-Sahai proofs are witness indistinguishable and so the request messages cannot be distinguished by environment $\mathcal{Z}$.

D Efficiency of the POT scheme

Let N_{eq} be the number of equations and m be the number of variables in a Groth-Sahai proof. Let N_{nl} be the number of non-linear equations and N_l be the number of linear equations. An equation is linear if either $\{\alpha_{q,i}\}_{q=1\ldots Q,i=1\ldots m} = 0$ or $\{\beta_{q,i}\}_{q=1\ldots Q,i=1\ldots m} = 0$. Let Q_a denote the number of linear pairings, where either $\{\alpha_{q,i}\}_{i=1\ldots m} = 0$ or $\{\beta_{q,i}\}_{i=1\ldots m} = 0$, and Q_b the number of quadratic pairings. When instantiated with the DLIN assumption, a Groth-Sahai proof that consists of N_{eq} pairing product equations has $9N_{nl} + 3N_l + 3m$ elements of $\mathbb{G}$. Calculating each group element requires a multiexponentiation. The verification requires the computation of $21N_{eq} + 3Q_a + 9Q_b$ pairings.[6]

The POT scheme consists of an initialization phase where both $\mathcal{B}$ and $\mathcal{V}$ send one message and k transfer phases where $\mathcal{V}$ sends a response upon receiving a request from $\mathcal{B}$. Therefore, it has $(k + 1)$-rounds and the communication cost is $O(N + k)$. crs consists of 23 elements of $\mathbb{G}$. Let $A = d^a$

[6]We make use of the fact that some of the pairings computed in the verification algorithm are of the form e(x,1).

be the upper bound of the deposit. In the initialization phase, $\mathcal{V}$ sends T, where each ciphertext has 12 elements of $\mathbb{G}$ and the public key consists of 2 elements plus 7 elements of the public key of the signature scheme plus $2 + 3d$ elements of the parameters of the range proof (where each signature consists of 3 elements and the public key has 2 elements). In total they sum $11 + 3d + 12N$ elements of $\mathbb{G}$. $\mathcal{B}$ sends an encryption of the account that consists of 5 elements of $\mathbb{G}$. In each transfer phase, $\mathcal{B}$ sends a request that has two elements of $\mathbb{G}$ and a proof pok_1, which consists of 5 pairing product equations $(N_l = 4, N_{nl} = 1, Q_a = 8, Q_b = 1)$ to prove possession of a multi-block signature, 4 equations $(N_l = 4, N_{nl} = 0, Q_a = 8, Q_b = 0)$ to prove that the request is correctly computed, 1 equation $(N_l = 1, N_{nl} = 0, Q_a = 3, Q_b = 0)$ to prove that $ac_i = ac_{i-1} - p_{\sigma_i}$, and $3a + 1$ equations $(N_l = 1+2a, N_{nl} = a, Q_a = 5a+1, Q_b = a)$ to prove that $ac_i \in [0..d^a)$. In total, pok_1 has parameters $(N_{eq} = 11 + 3a, m = 15 + 5a, N_l = 10 + 2a, N_{nl} = 1 + a, Q_a = 20 + 5a, Q_b = 1 + a)$ and thus comprises $84+30a$ elements of $\mathbb{G}$ and requires the computation of $300+87a$ pairings for verification. $\mathcal{V}$ sends back one element of $\mathbb{G}$ and a proof pok_2, which has parameters $(N_{eq} = 4, m = 3, N_l = 3, N_{nl} = 1, Q_a = 6, Q_b = 1)$ and thus comprises 27 elements of $\mathbb{G}$ and requires 111 pairings for verification.