Unlinkable Priced Oblivious Transfer with Rechargeable Wallets

Jan Camenisch[1], Maria Dubovitskaya[2,3], and Gregory Neven[1]

[1] IBM Research – Zurich
[2] IBM Russian Systems and Technology Laboratory
[3] National Research Nuclear University MEPhI, Russia

Abstract. We present the first truly unlinkable priced oblivious transfer protocol. Our protocol allows customers to buy database records while remaining fully anonymous, i.e., (1) the database does not learn who purchases a record, and cannot link purchases by the same customer; (2) the database does not learn which record is being purchased, nor the price of the record that is being purchased; (3) the customer can only obtain a single record per purchase, and cannot spend more than his account balance; (4) the database does not learn the customer's remaining balance. In our protocol customers keep track of their own balances, rather than leaving this to the database as done in previous protocols. Our priced oblivious transfer protocol is also the first to allow customers to (anonymously) recharge their balances. Finally, we prove our protocol secure in the standard model (i.e., without random oracles).

1 Introduction

Suppose you want to buy a digital item from a website that charges per purchased item, and that sells different items at different prices. You have reasons to believe, however, that the website is making a lucrative parallel business out of selling information about your shopping behavior to your competitors. For example, you may work for a pharmaceutical company and buy information about particular DNA genome sequences from a database, or you may work for a high-tech company and buy patents from a patent database. The list of purchased records from either of these databases certainly reveals precious information about your company's research strategies. How do you prevent the database from gathering information about your shopping behavior while still allowing the database to correctly charge you for the purchased items?

What we need here is a *priced oblivious transfer* (POT) protocol [1]. Here, customers load an initial amount of money into their pre-paid accounts, and can then start downloading records so that (1) the database does not learn which record is being purchased, nor the price of the record that is being purchased; (2) the customer can only obtain a single record per purchase, and cannot spend more than his account balance; (3) the database does not learn the customer's remaining balance. All known POT protocols require the database to maintain customer-specific state information across the different purchases by the same customer to keep track of his (encrypted) account balance. Each customer's transactions thereby necessarily become linkable. Thus, none of these protocols allows the customer to purchase records anonymously: even if an anonymous payment system is used to pre-charge the initial balance, the customer could be

at most pseudonymous, defeating the purpose of protecting the customer's privacy. For example, the database still learns the number of records bought by each customer, the time that these records were bought, and their average price. This clearly reveals information about the customer and might lead to identification of the customer or of the records she's buying. To overcome this, we further require that the POT satisfy that (4) the database does not learn any information about who purchases a record;

Existing POT protocols also lack a recharge functionality: Once a customer's balance does not contain enough credit to buy a record, but is still positive, the customer cannot use up the balance, but will have to open a new account/balance for further purchases.

In this paper, we propose the first truly anonymous priced oblivious transfer protocol with recharge functionality. Rather than having the database keep track of account balances, in our protocol the customers maintain their own balance. Of course, precautions have to be taken to ensure that they cannot tamper with their balance, or rewind it to a previous state. Furthermore, we offer a protocol that allows customers to recharge their balances. Lastly, we present an enhanced protocol where records are transferred using an optimistic fair exchange protocol [2, 3], thereby preventing a cheating database from decreasing a customer's wallet without sending the desired record.

In what follows we give an overview on how our protocol overcomes the various technical challenges posed by these demanding requirements.

1.1 Construction Overview

We consider a database where each record may have a different price. The database provider encrypts each record with a key that is derived from not only its index but also its price and then publishes the whole encrypted database.

To be able to access records, a customer first contacts the provider to create a new, empty wallet. Customers can load more money into their wallet at any time. The payment mechanism used to recharge customers' wallets is outside the scope of this paper; for full customer anonymity, we advise the use of an anonymous e-cash scheme.

When a customer wants to purchase a record with index σ with price p from the database, the provider and the customer essentially run a two party protocol at the end of which the customer will have obtained the decryption key for the record σ as well as an updated wallet being worth p monetary units less. This is done is such a way that the provider does not learn anything about σ or p. More precisely, we model wallets as one-time-use anonymous credentials with the worth of the wallet being encoded as an attribute. When the customer buys a record (or charges her wallet), she basically uses the credential and gets in exchange a new credential with the updated worth as attribute without the provider learning anything about the wallet's worth. The properties of one-time-use credentials ensure that a customer cannot buy records worth more than what she has (pre-)paid to the provider. We prove our protocol secure in the standard model (i.e., without random oracles).

1.2 Related Work

Relative to the enormous body of work that has appeared on oblivious transfer, only few priced oblivious transfer protocols have been proposed. The concept of POT was introduced by Aiello et al. [1] who present a scheme based on homomorphic encryption and

symmetrically private information retrieval (SPIR) [25]. The protocol by Tobias [28] is based on ElGamal, and a recent protocol by Rial et al. [30] builds on the OT protocol of [14]. The protocols of [1, 28] come only with heuristic security considerations, while that of [30] was proved secure in the universal composability (UC) model [20]. All three of these protocols share a common principle that the database maintains an encryption or commitment of each customer's balance that gets updated at each purchase. Purchases by the same customer are therefore necessarily linkable, as the database has to know which ciphertext or commitment to use. Neither of these protocols enables customers to recharge their wallets.

While by itself not being a POT protocol, the recent work by Coull et al. [22] could be cast into one. They propose an OT scheme where access to records is controlled by using a state graph. With each access a user transitions from one state to another, where the allowed records are defined by the possible transitions from the current state. One could implement a (fully anonymous) POT protocol by defining a separate state for each possible balance in a customer's wallet. The allowed transitions between states b and b' are those records with price exactly $b - b'$. When using this approach however, the size of the encrypted database is $O(b_{\max} \cdot N)$, where $b_{\max}$ is the maximum wallet balance and N is the number of records in the database, as opposed to $O(b_{\max} + N)$ in our scheme.

Our paper builds on ideas from recent work by Camenisch et al. [10] who extend the OT protocol of [14] with anonymous access control. One could in fact combine their and our ideas to achieve POT with access control. This would allow for price differentiation among customers while maintaining full customer privacy, e.g., offer a cheaper price to holders of a loyalty card.

2 Definition of UP-OT

2.1 Syntax

Let $\kappa \in \mathbb{N}$ be a security parameter and let ε be the empty string. All algorithms described here are probabilistic polynomial-time (PPT); we implicitly assume they all take an extra input 1^κ. A function $\nu : \mathbb{N} \to [0, 1]$ is *negligible* if for all $c \in \mathbb{N}$ there exists a $\kappa_c \in \mathbb{N}$ such that $\nu(\kappa) < \kappa^{-c}$ for all $\kappa > \kappa_c$.

An unlinkable priced oblivious transfer (UP-OT) scheme is parameterized by a security parameter $\kappa \in \mathbb{N}$, a maximum wallet balance $b_{\max} \in \mathbb{N}$ and a maximum record price $p_{\max} \leq b_{\max}$. We consider a setting with one database and one or more customers. A database consists of a list of N couples $((R_1, p_1), \ldots, (R_N, p_N))$, containing database records $R_1, \ldots, R_N \in \{0, 1\}^*$ and associated prices $p_1, \ldots, p_N \in \{0, \ldots, p_{\max}\}$. A UP-OT scheme is a tuple of polynomial-time algorithms and protocols $\mathcal{UP}\text{-}\mathcal{OT} = (\mathsf{DBSetup}, \mathsf{CreateWallet}, \mathsf{Recharge}, \mathsf{Purchase})$ run between customers $\mathsf{C}_1, \ldots, \mathsf{C}_M$ and a database provider DB in the following way:

– $\mathsf{DBSetup} : \mathsf{DB} : \big(DB = (R_i, p_i)_{i=1,\ldots,N}\big) \xrightarrow{\$} \big((pk_{\mathrm{DB}}, ER_1, \ldots, ER_N), sk_{\mathrm{DB}}\big)$
The database provider runs the randomized $\mathsf{DBSetup}$ algorithm to initiate a database containing records $R_1, \ldots, R_N$ with corresponding prices $p_1, \ldots, p_N$. It generates a pair of a secret and corresponding public key $(sk_{\mathrm{DB}}, pk_{\mathrm{DB}})$ for security parameter κ,

and uses it to encrypt the individual records. The encrypted database consists of the public key pk_{DB} and the encrypted records $ER_1, \ldots, ER_N$. The encrypted database is made available to all customers, e.g. by publishing it on a website or by distributing it on DVDs.[4] The database provider keeps the secret key sk_{DB} to himself.

– $\mathsf{CreateWallet} : \mathsf{DB} : (pk_{\mathrm{DB}}, sk_{\mathrm{DB}}) \rightarrow (\varepsilon)$; $\mathsf{C} : (pk_{\mathrm{DB}}) \rightarrow W_0$ or $\perp$

A customer creates an empty wallet with a zero balance, signed by the database provider, by engaging in the CreateWallet protocol with the database provider. The provider's public key pk_{DB} is a common input, the corresponding secret key sk_{DB} is a secret input to the provider. At the end of the protocol, the customer outputs an empty wallet W_0, or $\perp$ to indicate failure.

– $\mathsf{Recharge} : \mathsf{DB} : (pk_{\mathrm{DB}}, m, sk_{\mathrm{DB}}) \rightarrow (\varepsilon)$; $\mathsf{C} : (pk_{\mathrm{DB}}, m, W_i) \rightarrow W_{i+1}$ or $\perp$

When the customer wants to add money to her wallet W_i (which may or may not be her initial wallet W_0) she can engage in a Recharge protocol with the database provider. The database's public key pk_{DB} and the amount of money m that the customer wants to add to her balance are common inputs. The database's secret key sk_{DB} and the customer's current wallet W_i are private inputs to the database and the customer, respectively. At the end of the protocol, the customer either outputs the new wallet W_{i+1} or $\perp$ to indicate failure.

– $\mathsf{Purchase} : \mathsf{DB} : (pk_{\mathrm{DB}}, sk_{\mathrm{DB}}) \rightarrow (\varepsilon)$; $\mathsf{C} : (pk_{\mathrm{DB}}, \sigma, ER_\sigma, p_\sigma, W_i) \rightarrow (R_\sigma, W_{i+1})$ or $(\perp, W_{i+1})$ or $(R_\sigma, \perp)$ or $(\perp, \perp)$

To purchase a record from the database, a customer engages in a Purchase protocol with the database provider. The database's public key pk_{DB} is a common input. The customer has as a private input her selection index $\sigma \in \{1, \ldots, N\}$, the encrypted record ER_σ and its price p_σ, and her current wallet W_i. The database provider uses its secret key sk_{DB} as a private input. At the end of the protocol, the customer outputs the database record R_σ and an updated wallet W_{i+1}. An output containing $R_\sigma = \perp$ or $W_{i+1} = \perp$ indicates that the record transfer or the wallet update failed, respectively.

We assume that all communication links are private and anonymous, to that cheating customers cannot eavesdrop on honest customers' conversations, and so that the database does not know which customer he's interacting with.

2.2 Security

We define security of an UP-OT protocol through indistinguishability of a real-world and an ideal-world experiment reminiscent of the universal-composability (UC) framework [18, 19] and the reactive-systems security model [26, 29]. The definitions we give, however, do not entail all formalities necessary to fit either of these frameworks; our goal here is solely to prove security of our scheme.

We summarize the ideas underlying these models. In the real world the honest players and the adversary A who controls the dishonest players run cryptographic protocols

[4] We assume that each customer obtains a copy of the *entire* encrypted database. It is impossible to obtain our strong privacy requirements with a single database server without running into either computation or communication complexity that is linear in the database size. In this paper we focus on achieving our strong privacy notion while keeping the complexity of the purchase protocol constant, i.e., independent of N.

with each other. The environment $\mathcal{E}$ provides the inputs to the honest players and receives their outputs, and interacts arbitrarily with the adversary. In the ideal world, the players do not run any cryptographic protocols but interact through an ideal trusted party T. A (set of) cryptographic protocol(s) is said to securely implement a functionality if for every real-world adversary A and every environment $\mathcal{E}$ there exists an ideal-world simulator A' (controlling the same parties as A) such that $\mathcal{E}$ cannot distinguish with non-negligible probability whether it is run in the real world while interacting with A or whether it is run in the ideal world while interacting with A'.

THE REAL WORLD. We first describe how the real world algorithms presented in §2.1 are orchestrated when all participants are honest, i.e., honest real-world customers $\mathsf{C}_1, \ldots \mathsf{C}_M$ and an honest database DB. Parties controlled by the real-world adversary A can arbitrarily deviate from the behavior described below.

Upon receiving $(\texttt{initdb}, DB = (R_i, p_i)_{i=1,\ldots,N})$ from $\mathcal{E}$ the database generates its key pair and encrypts the records by running $((pk_{\mathrm{DB}}, EDB), sk_{\mathrm{DB}}) \xleftarrow{\$} \mathsf{DBSetup}(DB)$, and sends $(pk_{\mathrm{DB}}, EDB, (p_i)_{i=1,\ldots,N})$ to all customers $\mathsf{C}_1, \ldots, \mathsf{C}_M$.

Upon receiving $(\texttt{create\_wallet})$ from $\mathcal{E}$ customer C_j obtains an empty wallet by engaging in a $\mathsf{CreateWallet}$ protocol with the database provider on common input pk_{DB}. The provider uses his secret key sk_{DB} as a private input. At the end of the protocol, the customer obtains the empty wallet $W_0^{(j)}$ with zero balance or $\perp$ indicating failure. In the former case C_j outputs a bit 1 to $\mathcal{E}$, in the latter it outputs 0. DB does not return anything to the environment.

Upon receiving $(\texttt{recharge}, m)$ from $\mathcal{E}$ customer C_j engages in a $\mathsf{Recharge}$ protocol with DB on common input pk_{DB}, m, using sk_{DB} and $W_i^{(j)}$ as private inputs to DB and C_j, respectively. At the end of the protocol, C_j either obtains the new wallet $W_{i+1}^{(j)}$ or $\perp$. In the former case, it outputs a bit 1 to $\mathcal{E}$, in the latter it outputs 0. DB does not return anything to the environment.

Upon receiving $(\texttt{purchase}, \sigma_i)$ from $\mathcal{E}$, customer C_j engages in a $\mathsf{Purchase}$ protocol with DB on common input (pk_{DB}), on C_j's private input $\sigma_i, ER_{\sigma_i}, p_{\sigma_i}, W_i^{(j)}$, and on DB's private input sk_{DB}, until C_j obtains the record R_{σ_i} and a new wallet $W_{i+1}^{(j)}$. The customer returns two bits to the environment, the first indicating whether the record transfer succeeded (i.e., 0 if $R_{\sigma_i} = \perp$ and 1 otherwise), the second indicating whether the wallet update succeeded (i.e., 0 if $W_{i+1}^{(j)} = \perp$ and 1 otherwise). DB does not return anything to the environment.

THE IDEAL WORLD. In the ideal world all participants communicate through a trusted party T which implements the functionality of our protocol. We describe the behavior of T on the inputs of the ideal-world customers $\mathsf{C}'_1, \ldots, \mathsf{C}'_M$ and the ideal-world database DB'. The trusted party T maintains the database DB and an array $W[\cdot]$ to keep track of the balance in customer's wallets. Initially all entries are unspecified, i.e., $DB \leftarrow \varepsilon$ and $W[j] \leftarrow \varepsilon$ for $j = 1, \ldots, N$. The trusted party responds to queries from the different parties as follows.

Upon receiving $(\texttt{initdb}, (R_i, p_i)_{i=1,\ldots,N})$ from DB', T checks whether $0 \leq p_i \leq p_{\max}$ for $i = 1, \ldots, N$. If so, it sets $DB \leftarrow (R_i, p_i)_{i=1,\ldots,N}$ and sends a message $(\texttt{initdb}, N)$ to all customers.

Upon receiving $(\texttt{create\_wallet})$ from C'_j, T sends $(\texttt{create\_wallet})$ to DB' who sends back a bit b. If $b = 1$ then T sets $W[j] \leftarrow 0$ and sends 1 to C'_j; otherwise it simply sends 0 to C'_j.

Upon receiving $(\texttt{recharge}, m)$ from C'_j, T first checks that $W[j] \neq \varepsilon$ and $W[j] + m \leq b_{\max}$. If either of these checks fails, it sends back a bit 0 to C'_j, otherwise it proceeds as follows. T sends $(\texttt{recharge}, m)$ to DB' who sends back a bit b. If $b = 1$ then the T sets $W[j] \leftarrow W[j] + m$ and sends a bit 1 to C'_j; otherwise it simply sends 0 to C'_j.

Upon receiving $(\texttt{purchase}, \sigma_i)$ from C'_j, T proceeds as follows. If $W[j] < p_{\sigma_i}$ then T simply returns a pair $(\perp, 1)$ to C'_j. Otherwise, it sends a message $(\texttt{purchase})$ to DB', who sends back a pair of bits (b_1, b_2) indicating whether or not the record transfer and the wallet update succeeded. Party T sends a pair (R, b) back to C'_j that is composed as follows. If $b_1 = 1$ and $DB \neq \varepsilon$ then it sets $R \leftarrow R_{\sigma_i}$, otherwise it sets $R \leftarrow \perp$. If $b_2 = 1$ then T sets $W[j] \leftarrow W[j] - p_{\sigma_i}$ and $b \leftarrow 1$; else it sets $W[j] \leftarrow \varepsilon$ and $b \leftarrow 0$.

The honest ideal-world parties $\mathsf{C}'_1, \ldots, \mathsf{C}'_M, \mathsf{DB}$ simply relay inputs and outputs between the environment $\mathcal{E}$ and the trusted party T. Dishonest parties can deviate arbitrarily from this behavior.

Note that in the ideal world the database cannot tell which customer makes a purchase, which record she is querying, or what the price of this record is, therefore guaranteeing perfect customer privacy. At the same time, customers in the ideal world can only purchase records that they can afford, they can only obtain one per purchase, and even colluding customers cannot obtain records that they wouldn't have been able to afford individually, thereby guaranteeing perfect database security.

We point out that the ideal functionality of Rial et al. [30] models a single customer only, hence it cannot guarantee different customers to be indistinguishable to the database, and in fact, their protocol does not satisfy this property.

3 Preliminaries

Let $\mathsf{Pg}(1^\kappa)$ be a pairing group generator that on input 1^κ outputs descriptions of multiplicative groups $\mathbb{G}, \mathbb{G}_\mathrm{T}$ of prime order q where $q > 2^\kappa$. Let $\mathbb{G}^* = \mathbb{G} \setminus \{1\}$ and let $g \in \mathbb{G}^*$. The generated groups are such that there exists an admissible bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_\mathrm{T}$, meaning that (1) for all $a, b \in \mathbb{Z}_q$ it holds that $e(g^a, g^b) = e(g, g)^{ab}$; (2) $e(g, g) \neq 1$; and (3) the bilinear map is efficiently computable.

Definition 1 We say that the decision ℓ-bilinear Diffie-Hellman exponent (ℓ-BDHE) assumption holds in groups $\mathbb{G}, \mathbb{G}_\mathrm{T}$ of order $q > 2^\kappa$ if for all polynomial-time adversaries A the advantage $\mathbf{Adv}^{\mathrm{BDHE}}_{\mathbb{G},\mathbb{G}_T}(\kappa)$ of, given a tuple $(g, h, g^\alpha, \ldots, g^{\alpha^{\ell-1}}, g^{\alpha^{\ell+1}}, \ldots, g^{\alpha^{2\ell}}, S)$, to distinguish whether $S = e(g, h)^{\alpha^\ell}$ or $S \xleftarrow{\$} \mathbb{G}^*_\mathrm{T}$, is a negligible function in κ for $g, h \xleftarrow{\$} \mathbb{G}^*$ and $\alpha \xleftarrow{\$} \mathbb{Z}_q$. ∎

Definition 2 We say that the ℓ-strong Diffie-Hellman (ℓ-SDH) assumption [6] holds in group $\mathbb{G}$ of order $q > 2^\kappa$ if for all polynomial-time adversaries A the advantage $\mathbf{Adv}^{\mathrm{SDH}}_{\mathbb{G}}(\kappa) = \Pr\left[\mathsf{A}(g, g^x, \ldots, g^{x^\ell}) = (c, g^{1/(x+c)})\right]$ is a negligible function in κ, where $g \xleftarrow{\$} \mathbb{G}^*$ and $x, c \xleftarrow{\$} \mathbb{Z}_q$. ∎

3.1 Modified Boneh-Boyen Signatures

We use the following modification of the weakly-secure signature scheme by Boneh and Boyen [6]. The scheme uses a pairing generator Pg as defined above.

The signer's secret key is $(x_m, x_p) \xleftarrow{\$} \mathbb{Z}_q$, the corresponding public key is $(g, y_m = g^{x_m}, y_p = g^{x_p})$ where g is a random generator of $\mathbb{G}$. The signature on the tuple of messages (m, p) is $s \leftarrow g^{1/(x_m+m+x_p p)}$; verification is done by checking whether $e(s, y_m \cdot g^m \cdot y_p^p) = e(g, g)$ is true.

This signature scheme is the special case for $\ell = 2$ of the modified Boneh-Boyen signature scheme used by Camenisch et al. [10], who also show it to be unforgeable under weak chosen-message attack if the $(N + 1)$-SDH assumption holds.

3.2 Zero-Knowledge Proofs and Σ-Protocols

When referring to the zero-knowledge proofs, we will follow the notation introduced by Camenisch and Stadler [16] and formally defined by Camenisch, Kiayias, and Yung [11]. For instance, $PK\{(a, b, c) : y = g^a h^b \wedge \tilde{y} = \tilde{g}^a \tilde{h}^c\}$ denotes a "*zero-knowledge Proof of Knowledge of integers a, b, c such that $y = g^a h^b$ and $\tilde{y} = \tilde{g}^a \tilde{h}^c$ holds,*" where $y, g, h, \tilde{y}, \tilde{g}$, and $\tilde{h}$ are elements of some groups $G = \langle g \rangle = \langle h \rangle$ and $\tilde{G} = \langle \tilde{g} \rangle = \langle \tilde{h} \rangle$.

Given a protocol in this notation, it is straightforward to derive actual protocol implementing the proof. Indeed, the computational complexities of the proof protocol can be easily derived from this notation: for each term $y = g^a h^b$, the prover and the verifier have to perform an equivalent computation, and to transmit one group element and one response value for each exponent. We refer to Camenisch et al. [11] for details.

3.3 Wallet Signature Scheme

We use the signature scheme proposed and proved secure by Au et al. [4], which is based on the schemes of Camenisch and Lysyanskaya [12] and of Boneh et al. [7].

The signer's secret key is a random element $x \xleftarrow{\$} \mathbb{Z}_q$. The public key contains a number of random bases $g_1, h_0, \ldots, h_\ell, h_{\ell+1} \xleftarrow{\$} \mathbb{G}$, where $\ell \in \mathbb{N}$, and $y \leftarrow g_1^x$.

A signature on messages $m_0, \ldots, m_\ell \in \mathbb{Z}_q$ is a tuple (A, r, s) where $r, s \xleftarrow{\$} \mathbb{Z}_q$ are values chosen at random by the signer and $A = (g_1 h_0^{m_0} \cdots h_\ell^{m_\ell} h_{\ell+1}^r)^{1/(x+s)}$. Such a signature can be verified by checking whether $e(A, g_1^s y) = e(g_1 h_0^{m_0} \cdots h_\ell^{m_\ell} h_{\ell+1}^r, g_1)$.

Now assume that we are given a signature (A, r, s) on messages $m_0 \ldots, m_\ell \in \mathbb{Z}_q$ and want to prove that we indeed possess such a signature. To this end, we need to augment the public key with values $u, v \in \mathbb{G}$ such that $\log_{g_1} u$ and $\log_{g_1} v$ are not known. This can be done by choosing random values $t, t' \xleftarrow{\$} \mathbb{Z}_q$, computing $\tilde{A} = Au^t$, $B = v^t u^{t'}$ and executing the following proof of knowledge

$$PK\{(\alpha, \beta, s, t, t', m_0, \ldots, m_\ell, r) : B = v^t u^{t'} \ \wedge \ 1 = B^{-s} v^\alpha u^\beta \wedge \frac{e(\tilde{A}, y)}{e(g_1, g_1)} = e(\tilde{A}^{-s} u^\alpha h_{\ell+1}^r \textstyle\prod_{i=0}^{\ell} h_i^{m_i}, g_1) e(u, y)^t\} \ ,$$

where $\alpha = st$ and $\beta = st'$.

It was proved in [4] that the above signature is unforgeable under adaptively chosen message attack if Q-SDH assumption holds, where Q is the number of signature queries, and that the associated PoK is perfect honest-verifier zero-knowledge.

3.4 Set Membership Scheme

To prove that the customer's new balance after buying a record remains positive and is not more than maximum balance we use a signature-based set membership protocol suggested by Camenisch, Chaabouni and shelat [9].

They consider a zero-knowledge protocol which allow a prover to convince a verifier that a digitally committed value is a member of a given public set. This protocol has the verifier to produce signatures on the set elements, send them to the prover and then has the prover to show that he knows a signature (by the verifier) and the element he holds. Concretely, Camenisch et al. employed the weak signature scheme by Boneh and Boyen [6]. They prove that their protocol is a zero-knowledge argument of set membership for a set Φ, if the $|\Phi|$-SDH assumption holds.

4 Our UP-OT Construction

We now describe our scheme in detail. To issue wallets and update customers' balances, we employ the signature scheme presented in Section 3.3. To implement the oblivious transfer with anonymous payments we extend the OT protocol by Camenisch et al. [14]. We will also use a number of zero-knowledge proofs about discrete logarithms as described in Section 3.2.

Initial Setup. In Figure 1 we describe the setup procedure of the database provider, who also issues wallets to customers. Customers do not have their own setup procedure.

$\underline{\mathsf{DBSetup}\big(DB = (R_i, p_i)_{i=1,\ldots,N}\big) :}$

If $b_{\max} > 2^{\kappa-1}$ or $\exists i : p_i > p_{\max}$ then abort
$(\mathbb{G}, \mathbb{G}_{\mathrm{T}}, q) \xleftarrow{\$} \mathsf{Pg}(1^{\kappa})$; $g_t, h_t \xleftarrow{\$} {\mathbb{G}_{\mathrm{T}}}^*$; $g, h, g_1, h_1, h_2, h_3 \xleftarrow{\$} \mathbb{G}^*$; $x_{\mathrm{R}}, x_{\mathrm{p}}, x_{\mathrm{b}} \xleftarrow{\$} \mathbb{Z}_q$
$H \leftarrow e(g, h)$; $y_{\mathrm{R}} \leftarrow g^{x_{\mathrm{R}}}$; $y_{\mathrm{p}} \leftarrow g^{x_{\mathrm{p}}}$; $y_{\mathrm{b}} \leftarrow g_1^{x_{\mathrm{b}}}$
For $i = 1, \ldots, N$ do ; $E_i \leftarrow g^{\frac{1}{x_{\mathrm{R}}+i+x_{\mathrm{p}}\cdot p_i}}$; $F_i \leftarrow e(h, E_i) \cdot R_i$; $ER_i \leftarrow (E_i, F_i)$
For $i = 0, \ldots, b_{max}$ do $y_{\mathrm{b}}^{(i)} \leftarrow g^{1/(x_{\mathrm{b}}+i)}$
$sk_{\mathrm{DB}} \leftarrow (h, x_{\mathrm{R}}, x_{\mathrm{p}}, x_{\mathrm{b}})$; $pk_{\mathrm{DB}} \leftarrow (g, H, g_1, h_1, h_2, h_3, y_{\mathrm{R}}, y_{\mathrm{p}}, y_{\mathrm{b}}, y_{\mathrm{b}}^{(0)}, \ldots, y_{\mathrm{b}}^{(i)})$
Return $\big(pk_{\mathrm{DB}}, sk_{\mathrm{DB}}, (ER_1, \ldots, ER_N)\big)$

Fig. 1. Database Setup algorithm

The database provider runs the randomized DBSetup algorithm to initiate a database containing records $R_1, \ldots, R_N$ with corresponding prices $p_1, \ldots, p_N$. Then it generates a pairing group of prime order q for security parameter κ, a number of random generators, and three secret keys x_{R}, x_{p}, and x_{b} with corresponding public keys y_{R}, y_{p}, and y_{b}. Intuitively, x_{R} is the seed used to generate randomness to encrypt the records, x_{p} securely links prices to records, and x_{b} authenticates the balance in customers' wallets.

The database provider encrypts each record R_i with its own key to a ciphertext (E_i, F_i). These keys are signatures on the index i and the price p_i of the record made with the database provider's secret keys x_{R} and x_{p}. The pairs (E_i, F_i) can be seen as an ElGamal encryption [24] in $\mathbb{G}_{\mathrm{T}}$ of the record R_i under the public key H. But

instead of using random elements from $\mathbb{G}_{\mathrm{T}}$ as the first component, our protocol uses verifiably random [23] values $E_i = g^{\frac{1}{x_{\mathrm{R}}+i+x_{\mathrm{P}}\cdot p_i}}$. It is this verifiability that during the purchase protocol allows the database to check that the customer is indeed asking for the decryption key of a single record with a price that is within the customer's current budget.

Let $p_{max} \leq b_{\max} < 2^{\kappa-1} < q/2$ be the maximal balance that can be stored in a customer's wallet. To prove that the customer's new balance after buying a record remains positive and is not more than maximum balance we use a signature-based set membership protocol [9]. Here the set contains all possible balances from the customer's wallet $\{0, \ldots, b_{\max}\}$. So for each possible balance $0 \leq i \leq b_{\max}$ the database provider uses x_{b} to compute a signature $\{y_{\mathrm{b}}^{(i)}\}$. These values are included in the database's public key; they will be used by the customer to prove that her balance remains positive after subtracting the price of the purchased record.

The encrypted database consists of a public key pk_{DB} and the encrypted records $ER_1, \ldots, ER_N$. It is made available to all customers, e.g. by publishing it on a website or by distributing it on DVDs. The server keeps the database secret key sk_{DB} to itself.

Obtaining wallets. Before purchasing any records, customers first need to create an empty wallet and then charge it with money. To create a wallet, the customer runs the CreateWallet protocol with the database provider depicted in Figure 2.

The database provider's public key pk_{DB} is a common input. The database provider has his secret key sk_{DB} as a private input. At the end of the protocol, the customer obtains a wallet $W_0 = (A_0, r_0, s_0, n_0, b_0)$ signed by the database provider. Here, (A_0, r_0, s_0) is essentially a signature as per the scheme of Section 3.3 of a serial number n_0 chosen by the customer and the initial balance of the wallet $b_0 = 0$. Next, the customer verifies the wallet's signature and outputs W_0 if the check is successful.

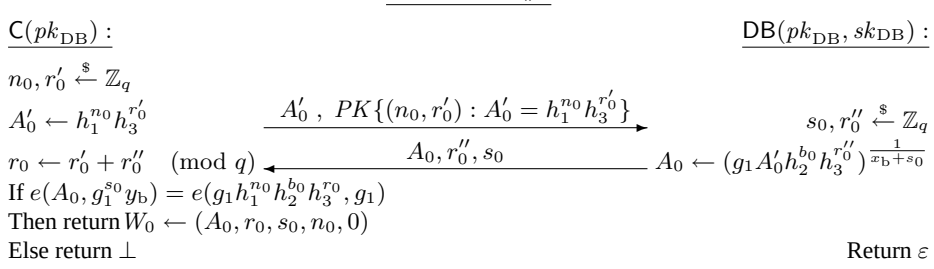

Fig. 2. Create wallet protocol

Recharge protocol. Customers can recharge the balance of their wallets by engaging in a Recharge protocol (Figure 3) with the database server. Doing so does not reveal the remaining balance in the wallet, nor whether this is a freshly created wallet or an updated wallet obtained after purchasing a record. The common inputs are the database provider's public key pk_{DB} and the amount of money m that the customer wants to add to her balance. The database's secret key sk_{DB} and the customer's current wallet W_i are private inputs to the database and the customer, respectively.

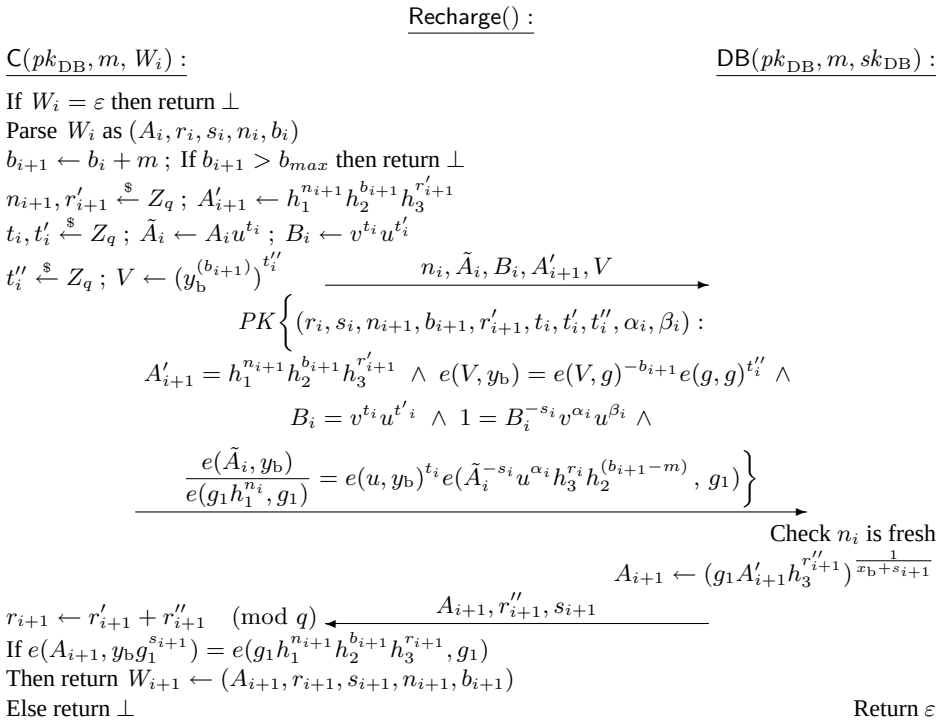

Fig. 3. Recharge protocol

If the customer already obtained a wallet earlier (her state is not empty), she updates her balance to $b_{i+1} = b_i + m$ and generates a fresh serial number n_{i+1} and a randomizer for the new wallet. Then she chooses from the set of database signatures $y_{\mathrm{b}}^{(1)}, \ldots, y_{\mathrm{b}}^{(b_{\max})}$ of possible balances the signature corresponding to her new balance and blinds it as $V = (y_{\mathrm{b}}^{(b_{i+1})})^{t''_i}$. This allows her to next prove that her new balance b_{i+1} is positive and is less than b_{max} with the set membership scheme from [9]. The customer further proves that she correctly increased her balance by the amount m being deposited. The database provider checks this proof and if the serial number n_i is fresh, i.e., whether it previously saw the number n_i. If not, then the database decides that the customer is trying to overspend and aborts. Otherwise, if the database provider accepts the proof, it signs the customer's new wallet with updated balance and sends it to the customer. The customer checks the validity of the signature on her new wallet, and if it verifies correctly, outputs an updated state containing the new wallet W_{i+1}.

Purchase protocol. When the customer wants to purchase a record from the database, she engages in a Purchase protocol (Figure 4) with the database server. The only common input is the database's public key pk_{DB}. The customer has as a private input her selection index $\sigma \in \{1, \ldots, N\}$, the encrypted record ER_σ and its price p_σ, and her current wallet W_i. The database provider uses its secret key sk_{DB} as a private input.

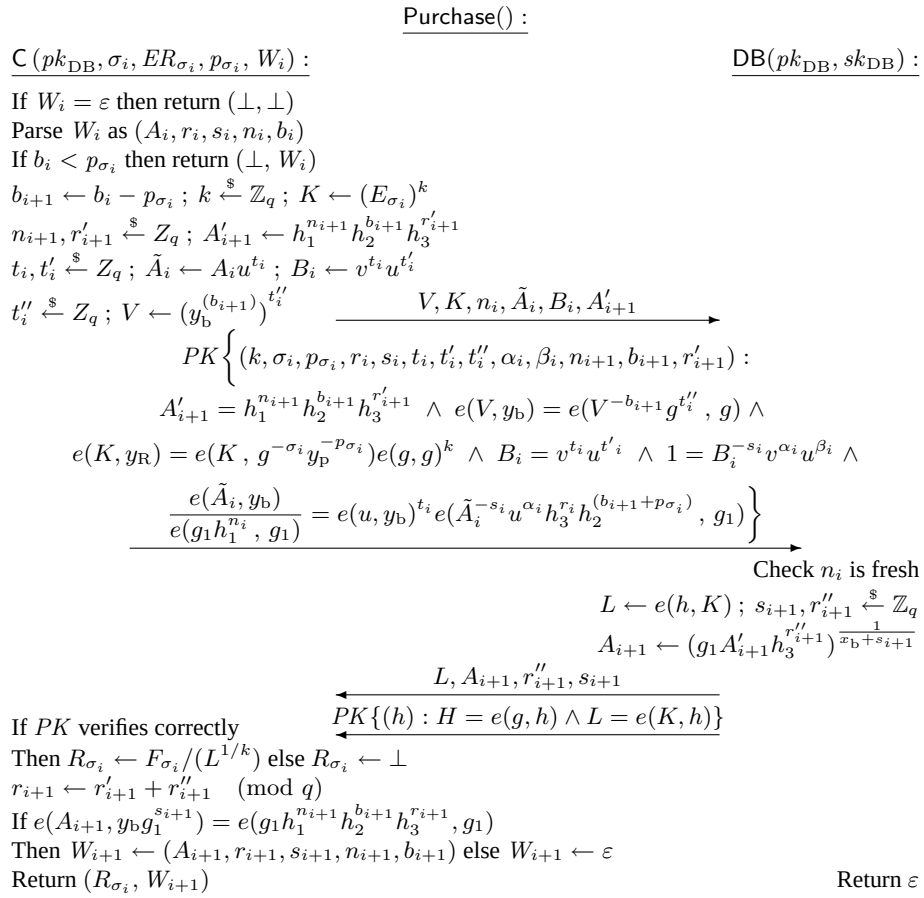

Fig. 4. Purchase protocol

The customer blinds the first part of the chosen encrypted record E_{σ_i} and sends this blinded version K to the database. Note that E_{σ_i} is derived from the database provider's secret key, the index and the price of the record. Next, the customer updates her balance to $b_{i+1} = b_i - p_{\sigma_i}$, generates a fresh serial number n_{i+1} and a randomizer for the new wallet. Then she chooses from the set of database signatures $y_{\mathrm{b}}^{(0)}, \ldots, y_{\mathrm{b}}^{(b_{\max})}$ of possible balances the signature corresponding to her new balance and blinds it as $V = (y_{\mathrm{b}}^{(b_{i+1})})^{t''_i}$. This allows her to prove that her new balance b_{i+1} is positive employing the set membership scheme from [9]. The customer further proves that K is correctly formed as a blinding of some E_{σ_i} and that she correctly reduced her balance by the price of requested record. The database provider checks if the serial number n_i is fresh, i.e., whether it previously saw the number n_i. If not, then the database decides that the customer is trying to double-spend and aborts. Otherwise, if the database provider accepts the proof, it computes L from h and K, sends L to the customer, and executes a proof of knowledge of the database secret key h and that L was computed correctly. In our security proof, this zero-knowledge proof will enable us to extract h and decrypt

the contents of the database. Also database provider signs the customer's new wallet with updated balance and sends it to the customer. The customer checks the validity of the zero-knowledge proof and of the signature on her new wallet. $\perp$ as the record; if the wallet signature is invalid, then it returns ε as the new wallet; if all goes correctly, she outputs the record R_σ and the new wallet W_{i+1}.

The protocol is easily seen to be correct by observing that $L = e(h, E_{\sigma_i})^k$, so therefore $F_\sigma / L^{1/k} = R_{\sigma_i}$.

Notice that $PK\{(h) : H = e(g,h) \wedge L = e(K,h)\}$ can be realized in the standard way as $e(g,\cdot)$ is a group (one-way) homomorphism that maps $\mathbb{G}$ onto $\mathbb{G}_{\mathrm{T}}$.

We finally remark that the database has to calculate a signature of every element in the set of all possible balances in the customer's wallet $\{0, \ldots, b_{max}\}$ and encrypt all records $(1, \ldots, N)$ only once at the setup phase, and the customer has to download the entire encrypted database and balance signatures only once as well. So the communication and computation complexity of the protocol depends on a cardinality of a set of all possible balances in the customer's wallet and a number of the records in the database only at the setup phase. The rest parts of the protocol (create wallet, recharge and purchase), however, require only a constant number of group elements to be sent and a constant number of exponentiations and pairings to be calculated.

5 Security Analysis

The security of our protocol is analyzed by proving indistinguishability between adversary actions in the real protocol and in an ideal scenario that is secure by definition. Given a real-world adversary A, we construct an ideal-world adversary A' such that no environment $\mathcal{E}$ can distinguish whether it is interacting with A or A'.

Theorem 3 *If the $(N+2)$-BDHE assumption holds in $\mathbb{G}, \mathbb{G}_T$ and the $max(q_{\mathrm{W}}, b_{\max}+1, N+1)$-SDH assumption holds in $\mathbb{G}$,then the $\mathcal{UP}$-$\mathcal{OT}$ protocol depicted in Figures 1–4 securely implements the UP-OT functionality, where N is the number of database records, $b_{\max}$ is the maximum possible balance in a customer's wallet, and q_{W} is the number of created wallets.* ∎

We prove the theorem in two steps: first, we prove the case where the database is corrupted, and next, we prove the case where a subset of the customers are corrupted. We do not consider the cases where all parties are honest and where all parties are dishonest as these cases have no practical interest. By lack of space, we only provide sketches of the construction of the ideal-world adversary A' below. A detailed proof is available in the full version of this paper.

Corrupted database. We first prove that for all environments $\mathcal{E}$ and all real-world adversaries A controlling the database there exists an ideal-world adversary A' such that $\mathcal{E}$ can distinguish the real world from the ideal world with probability at most $2^{-\kappa}$.

Since the adversary can always simulate additional customers himself, we can simplify the setting to a single honest customer C. We construct an ideal-world adversary A' that plays the role of the database and that runs the real-world adversary A as a subroutine as follows.

A' simply relays all messages between the environment $\mathcal{E}$ and A. It runs A to obtain the database's public key pk_{DB} and the encrypted database $EDB = (pk_{\mathrm{DB}}, (E_1, F_1), \ldots, (E_N, F_N))$.

Upon receiving a message $(\texttt{create\_wallet})$ from T, it executes the customer's side of the CreateWallet protocol with A. If the obtained wallet is valid, A' returns $b = 1$ to T, otherwise it returns $b = 0$.

Upon receiving $(\texttt{recharge}, m)$ from T, A' executes the customer's side of the Recharge protocol for amount m with A, but replaces the value V with a random element from $\mathbb{G}$ and simulates the PK protocol. If the protocol returns a valid wallet, then A' returns $b = 1$ to T; if the protocol returns $\perp$ then A' returns $b = 0$.

At the first purchase message from T, A' simulates an honest user querying for R_1, but replaces V with a random value from $\mathbb{G}$ and simulates the proof of knowledge. Then it extracts h from A in the last proof of knowledge, uses it to decrypt R_i as $F_i/e(h, E_i)$ for $i = 1, \ldots, N$ and sends $(\texttt{initdb}, (R_i, p_i)_{i=1,\ldots,N})$ to T. A' sends a pair of bits (b_1, b_2) back to T depending whether the obtained record is valid ($b_1 = 1$) or not ($b_1 = 0$) and whether the updated wallet is valid ($b_2 = 1$) or not ($b_2 = 0$).

Corrupted customers. Next, we prove that for all environments $\mathcal{E}$ and all real-world adversaries A controlling some of the customers, there exists an ideal-world adversary A' such that $\mathcal{E}$ can distinguish the real from the ideal world with probability at most

$$2^{-\kappa} \cdot Q + \ \mathbf{Adv}_{\mathbb{G}}^{\text{q-SDH}}(\kappa) + (N+1) \cdot \mathbf{Adv}_{\mathbb{G},\mathbb{G}_{\mathrm{T}}}^{(\mathrm{N}+2)\text{-BDHE}}(\kappa),$$

where Q is the total number of create wallet, recharge, and purchase queries, $q = \max(q_{\mathrm{W}}, b_{\max} + 1, N + 1), q_{\mathrm{W}}$ is the total number of create wallet queries, $b_{\max}$ is the maximum possible balance in a customer's wallet, and N is the number of records in the database.

Since the UP-OT functionality prevents colluding customers from pooling their wallets we have to consider multiple customers here, some of which are corrupted, and some of which are honest. Given a real-world adversary A controlling some of the customers, we construct an ideal-world adversary A' controlling the same customers as follows.

A' simply relays all communication between the environment $\mathcal{E}$ and A. Upon receiving $(\mathsf{DB}_{\mathsf{I}}, N)$ from T, A' creates an encrypted database (EDB, sk_{DB}) encrypting N random records, meaning that $F_1, \ldots, F_N$ are random group elements of $\mathbb{G}_{\mathrm{T}}$.

To simulate A's CreateWallet, Recharge, and Purchase protocol queries, A' plays the role of a real-world database by extracting from A the wallet signatures (A_i, r_i, s_i), record signatures E_{σ_i} and range proof signatures $y_{\mathrm{b}}^{(i)}$ that it uses in the zero-knowledge proofs, and aborting if any type of forged signature is detected. When a cheating customer C_j (controlled by A) makes a CreateWallet query, A' simply sends a message $(\texttt{create\_wallet}, j)$ to T; when C_j makes a Recharge query for amount m, A' sends a message $(\texttt{recharge}, j, m)$ to T. These never cause C'_j's balance to exceed $b_{\max}$, because that would imply a forgery in the range proof signature, which we ruled out above.

When a cheating customer C_j makes a Purchase query, A' extracts the index of the record being purchased σ_i and the exponent k, and sends $(\texttt{purchase}, j, \sigma_i)$ to T to obtain the record R_{σ_i}. Note that C'_j's balance is always sufficient to purchase R_{σ_i}, because otherwise A would have forged one of the signature schemes, which we ruled out above. Next, A' computes $L \leftarrow (F_{\sigma_i}/R_{\sigma_i})^k$ and simulates the zero-knowledge proof as in the simulation above.

6 A Fair Purchase Protocol

In the purchase protocol of our basic scheme, the customer has to spend her current wallet (i.e., reveal the serial number n_i) before obtaining the decryption key to the purchased record. A malicious database could abuse of this situation by aborting the transaction after the wallet was spent, thereby leaving the customer with a spent wallet and without the record that she wanted.

In this section we sketch how to strengthen our purchase protocol against these type of attacks. Essentially, we let the customer and the database engage in an optimistic fair exchange [2, 3] of the serial number n_i against the record decryption key L and updated wallet $(A_{i+1}, r''_{i+1}, s_{i+1})$. The customer and the database proceed as in the simple purchase protocol, except that instead of sending n_i in the clear the customer sends a commitment $cmt = \mathsf{Commit}(n_i)$ to the database, and performs the proof of knowledge $PK\{(k, \sigma_i, \ldots)\}$ based on this commitment. If the proof is accepted, the server sends back a verifiable encryption [15] of the decryption key and the new wallet $(L, A_{i+1}, r''_{i+1}, s_{i+1})$ under the trusted third party's public key, and performs the proof of knowledge $PK\{(h) : \ldots\}$ based on this verifiable encryption. The database also computes the hash h of all previously revealed serial numbers and provides the customer with a signature on cmt, h. Only after receiving the signature and verifying the PK does the customer reveal the serial number n_i to the database. The database checks if it is fresh serial number, and if so opens $L, A_{i+1}, r''_{i+1}, s_{i+1}$ to the customer.

If the database tries to cheat by not sending the decryption key or the new wallet, then the customer can take n_i, the verifiable encryption and the database's signature to the TTP. The TTP will contact the database to ask for the list of serial numbers that were included in the computation of h and checks whether n_i appears in this list. If not, then the database decrypts $L, A_{i+1}, r''_{i+1}, s_{i+1}$ for the customer. If it does, the TTP decides that the customer's complaint was unjustified and does nothing.

If the customer tries to cheat by reusing an old wallet or not opening the commitment cmt, then the database simply doesn't reveal $L, A_{i+1}, r''_{i+1}, s_{i+1}$ to the customer.

7 Acknowledgements

This work was supported in part by the European Community through the Seventh Framework Programme (FP7/2007-2013) project PrimeLife (grant agreement no. 216483) and through the ICT programme project ECRYPT II (contract ICT-2007-216676).

References

1. W. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, 119–135. Springer, 2001.
2. N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In *ACM CCS 97*, 7–17. ACM Press, 1997.
3. N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE JSAC*, 18(4):593–610, 2000.
4. M. H. Au, W. Susilo, and Y. Mu. Constant-size dynamic k-TAA. In R. D. Prisco and M. Yung, editors, *SCN 06*, volume 4116 of *LNCS*, 111–125. Springer, 2006.
5. M. Bellare and O. Goldreich. On defining proofs of knowledge. In E. F. Brickell, editor, *CRYPTO '92*, volume 740 of *LNCS*, 390–420. Springer, 1993.

6. D. Boneh and X. Boyen. Short signatures without random oracles. In C. Cachin and J. Camenisch, editors, *EUROCRYPT'04*, volume 3027 of *LNCS*, 56–73. Springer, 2004.
7. D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In M. K. Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS* , 41–55. Springer, 2004.
8. S. Brands. Rapid demonstration of linear relations connected by boolean operators. In W. Fumy, editor, *EUROCRYPT'97* , volume 1233 of *LNCS* , 318–333. Springer, 1997.
9. J. Camenisch, R. Chaabouni, and a. shelat. Efficient protocols for set membership and range proofs. In J. Pieprzyk, editor, *ASIACRYPT'08*,vol. 5350 of *LNCS*,234–252.Springer, 2008.
10. J. Camenisch, M. Dubovitskaya, and G. Neven. Oblivious transfer with access control. In E. Al-Shaer, S. Jha, and A. Keromytis, editors, *In ACM CCS 09*, 131-140. ACM Press, 2009
11. J. Camenisch, A. Kiayias, and M. Yung. On the portability of generalized schnorr proofs. In A. Joux, editor, *EUROCRYPT 2009*, LNCS . Springer, 2009.
12. J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In M. K. Franklin, editor, *Advances in Cryptology — CRYPTO 2004*, volume 3152 of *LNCS* , 56–72. Springer, 2004.
13. J. Camenisch and M. Michels. Proving in zero-knowledge that a number n is the product of two safe primes. In J. Stern, editor,*CRYPTO'99* , vol.1592 of *LNCS*,107–122.Springer,1999.
14. J. Camenisch, G. Neven, and a. shelat. Simulatable adaptive oblivious transfer. In M. Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, 573–590. Springer, 2007.
15. J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In D. Boneh, editor, *CRYPTO'03*, volume 2729 of *LNCS*, 126–144. Springer, 2003.
16. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In B. Kaliski, editor, *CRYPTO'97* , volume 1296 of *LNCS* , 410–424. Springer, 1997.
17. J. L. Camenisch. *Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem*. PhD thesis, ETH Zürich, 1998. Diss. ETH No. 12520.
18. R. Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, Weizmann Institute of Science, Rehovot 76100, Israel, June 1995.
19. R. Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
20. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, 136–145. IEEE Computer Society Press, 2001.
21. D. Chaum and T. P. Pedersen. Wallet databases with observers. In E. F. Brickell, editor, *CRYPTO'92* , volume 740 of *LNCS* , 89–105. Springer, 1993.
22. S. Coull, M. Green, and S. Hohenberger. Controlling access to an oblivious database using stateful anonymous credentials. Cryptology ePrint Archive, Report 2008/474, 2008.
23. Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. In S. Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, 416–431. Springer, 2005.
24. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.
25. Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. In *30th ACM STOC*, 151–160. ACM Press, 1998.
26. B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM CCS* , 245–254. ACM press, Nov. 2000.
27. C. P. Schnorr. Efficient signature generation for smart cards. *Journal of Cryptology*, 4(3):239–252, 1991.
28. C. Tobias. Practical oblivious transfer protocols. In F. A. P. Petitcolas, editor, IH 2002, volume 2578 of *LNCS*, 415–426. Springer, 2003.
29. B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE SRSP, 184–200. IEEE Press,* 2001.
30. A. Rial, M. Kohlweiss, and B. Preneel. Universally composable adaptive priced oblivious transfer. In H. Shacham and B. Waters, editors, *Pairing 2009*, LNCS, 24. Springer, 2009.