Oblivious Homomorphic Encryption

Osman Biçer and Christian Tschudin
University of Basel
osman.bicer@unibas.ch, christian.tschudin@unibas.ch

Oct 31, 2023

Abstract

In this paper, we introduce *Oblivious Homomorphic Encryption (OHE)* which provably separates the computation spaces of multiple clients of a fully homomorphic encryption (FHE) service while keeping the evaluator blind about whom a result belongs. We justify the importance of this strict *isolation* property of OHE by showing an attack on a recently proposed key-private cryptocurrency scheme. Our two OHE constructions are based on a puncturing function where the evaluator can effectively *mask* ciphertexts from rogue and potentially colluding clients. In the first construction, we show that this can be implemented via an FHE scheme (with key privacy and weak wrong-key decryption properties) plus an anonymous commitment scheme. In the second construction, for flexibility of primitive choice, we relax the FHE scheme as two separate encryption schemes: a standard FHE scheme and an encryption scheme with key privacy and weak wrong-key decryption. OHE can be used to provide provable anonymity to cloud applications, to single server implementations of anonymous messaging as well as to account-based cryptocurrencies.

Keywords— fully homomorphic encryption, key privacy, cloud data privacy, oblivious message retrieval, anonymous cryptocurrencies

1 Introduction

Suppose that you would like to run a cloud server where multiple clients upload their databases in encrypted form and query them for arbitrary computations on their content, including updating their databases, all being executed by your server. Your marketing pitch involves the claim of improved privacy to the clients such that even you are unable to deduce any information about their data, the queries that they make, and even which of the clients makes a particular query. This service would not only be useful for improved privacy but also for immunity to deplatforming, i.e. the possibility that a provider selectively denies its service for certain operations of selected clients. In an applicable setting to this scenario, the following criteria should be met:

- As the queries are expected to remain anonymous, the server processes each query on all the encrypted databases and produces, for each query, an encrypted result for each database. This query processing can be fully parallelized such that the completion time depends only on the provider's scaling.
- In case of an update request, all databases are rewritten at the ciphertext level. However, at the plaintext level (to which the provider has no access), only the specific client's database is changed. This means that your system needs a "masking" procedure that "isolates" the query, i.e. it neutralizes the request for all databases but the one it is intended for.

With these two principles (uniform processing of a query over all databases, uniform updating with requestor-specific masking), the anonymity claim holds if we can show that results are not linkable to a

public key (which would link the result to a client) and that updates do not leak which database was changed nor that a wrong database's content can be changed. Further, this update limitation is strict, meaning that even if multiple clients conjoin their queries, they cannot change others' content (this is a property that prevents account sharing). The purpose of this paper is to show that such a system can be built. Based on the techniques used, we call this *Oblivious Homomorphic Encryption (OHE).*

1.1 Potential Applications

There are plenty of private computation applications where additional privacy and isolation would be of importance, such as private databases [EZ19, BKK+21, WBNM22], searchable encryption [KT19, DPPS20], outsourced cryptographic computations [CLT14, QYL+22], machine learning and data mining [CXLC14, LC10], image processing [HWW+16, WWH+16], outsourcing biometric recognition data [WHR+15], anonymous messaging [WCGFJ12, CGBM15, Lun18] and privacy-preserving cryptocurrencies (in particular account based ones, e.g. Ethereum [Eth], Filecoin [Fil], Ripple [Rip], and GOC-ledger [Lav23]).

Privacy of Outsourced Computation. We wish to make indistinguishable requests so as to prevent correlation attacks based on request frequency while preventing conjoining queries by collusion. Outsourcing computations from clients to a server, although protected by plain FHE, does not prevent leakage of metadata, i.e. which client contacts a server, for which task, and how often. This puts the server in a position to selectively reject tasks or to sell said information. Just making the compute service anonymous is not sufficient as this would open the door for denial-of-service attacks against the server, using bogus public keys. Our OHE scheme, with the help of zero-knowledge and digital signatures, enables us to simultaneously address the clients' as well as the service provider's concerns.

Anonymous Messaging. In recent years private messaging [WCGFJ12, CGBM15, Lun18] has been a center of attention. Although one can obtain anonymous messaging with a straightforward application of anonymous encryption over a broadcast channel, the main challenge has been obliviously filtering the messages to a particular receiver (by a message delivery server) and handing only the messages pertinent to that user. [LT22] provides three separate oblivious message retrieval protocols as solutions to the problem. The first one "OMR1" uses a key-private FHE scheme and wrong-key decryption (a strong assumption) but is prone to DoS attacks by generating malicious "clues" as the authors of [LT22] acknowledge. In this work, we show that an oblivious message retrieval scheme can be obtained from OHE that is not susceptible to those attacks.

Privacy of Cryptocurrencies. Privacy has been considered of prime importance in cryptocurrencies. Some privacy-preserving payment schemes like QuisQuis [FMMO19], Zether [BAZB19], and anonymous Zether [Dia21] have been proposed for account-based cryptocurrencies, but achieving full anonymity on these schemes has only been a concern of [MS23]. Unfortunately, their privacy solution comes at the price of a double-spending vulnerability, as we show in this paper. Using OHE we are able to achieve the desired privacy for an account-based cryptocurrency scheme without risking the security of the payments.

1.2 Limitation of FHE for Privacy and Isolation

Fully homomorphic encryption (FHE) schemes [Gen09, SV10, DGHV10, BV11, BGV12, FV12, LATV12, Bra12, BLLN13, GSW13, BV14, DM15, CGGI16] have been proposed to allow a client to outsource the evaluation of a boolean circuit C of arbitrary length on a private input bit string x to an evaluator which will return the ciphertext of the output data $y = C(x)$. The main focus of FHE research has been on improving their efficiency, and to the best of our knowledge, FHE schemes have so far not been proven or designed to have "key privacy" [BBDP01, LPQ12, BM20, HLH+22], which is a property that prevents detection of the encryption key given a ciphertext. Further, even if some FHE schemes would be found to be key-private, the evaluator still needs to know the public key that encrypts a message as key privacy does not itself imply oblivious evaluation.

Recently, *wrong-key decryption* has been identified in [LT22] as an important property for oblivious evaluation. This property states that the decryption of a ciphertext with a secret key unrelated to the

public key used for its encryption must return random bits. FHE schemes with key privacy and wrong-key decryption come close to our expectations. However, they are not sufficient to provide what we call the *isolation property*. As context, we point to two approaches similar to our OHE: the FHE-based full-privacy cryptocurrency scheme PriFHEte of [MS23] and the FHE-based oblivious message retrieval OMR1 scheme of [LT22]. Regarding the former, we show a double-spending attack where it is possible to maliciously generate a ciphertext (of a transaction) that can pass masking of the evaluator under more than one public key, hence enabling a user to send money to multiple clients in total amount more than the one being deducted from him. Regarding the latter, the same issue exists and can be exploited for a denial of service (DoS) attack, which the authors in [LT22] acknowledge. What is needed, and what OHE brings out, is provable *isolation*, which is the property that each FHE key creates its own (encrypted) computation space without any crosstalk.

1.3 Our Contribution

In this paper we propose a novel and useful FHE scheme, i.e. the OHE scheme, with key privacy and isolation properties. Specifically, its isolation property is novel in the realm of FHE research and non-trivial to achieve in the presence of wrong-key decryption (a property necessitated by allowing computations on ciphertexts encrypted under different keys). More concretely, the original contributions of this paper are listed as follows:

1. In Section 5, we provide two OHE constructions: OHE1 from an FHE scheme with key privacy and "weak wrong-key decryption property" (a relaxed form of the wrong-key decryption of [LT22]) and an anonymous commitment scheme; and OHE2 from an FHE scheme, a weak wrong-key encryption (WWKE) scheme (i.e. relaxed form of wrong-key encryption of [LT22, MS23]), and an anonymous commitment scheme.
2. In Section 4, we show a double-spending attack on the FHE-based cryptocurrency system PriFHEte of [MS23] where the lack of isolation leads to the possibility that a user can generate transactions that increase other users' balances more than the amount deducted from her account. This attack shows the importance of the isolation property that we achieve with OHE.
3. In Section 6, we propose applications of our OHE scheme in common privacy scenarios:
    - **Improved Privacy in Cloud Computing.** Using our OHE scheme, a non-interactive zero-knowledge proof (NIZK) scheme, and a digital signature enables us to simultaneously address the cloud clients' concern that their data may be leaked as well as the cloud service provider's concern that clients may collude to generate combined queries, potentially rendering the service unprofitable.
    - **DoS-Resistant Oblivious Message Retrieval.** We show that an oblivious message retrieval scheme can be obtained from only OHE with relaxed choices of underlying schemes unlike "OMR2" and "OMR3" of [LT22], whose security can only be based on the learning with errors (LWE) assumption. Our construction is similar to the FHE-based OMR1 of [LT22]. However, unlike OMR1, our proposal is not susceptible to DoS attacks by generating "clues" maliciously [LT22].
    - **Optimized Cloud Computation.** We show how our OMR scheme can be used to optimize the communication complexity of the first application scenario such that clients only receive their results, without loss of privacy. This proposal combines our OMR proposal with our private cloud solution, both of which are mentioned above.
    - **Full Privacy of Account-Based Cryptocurrencies.** By combining OHE with the replay protection of [MS23], we are able to achieve the full privacy for an account-based cryptocurrency scheme (anonymity of the sender and the receiver, and privacy of the amount being sent in a transaction) – a solution that could be easily extended to also cover the privacy of executing smart contracts. Our construction is not susceptible to the double-spending attack on [MS23] that we show in Section 4.

1.4 Summary of Techniques

We start by relaxing the wrong-key decryption property of [LT22] as weak wrong-key decryption, which in a nutshell means that ciphertexts, when they are decrypted with different keys than the ones they are generated from, return a valid plaintext (i.e. a plaintext with the same size as the original one) with 1 minus negligible probability. The weak form of this property is easier to prove compared to the original wrong-key decryption property.

OHE1. We construct the first OHE scheme OHE1 from an FHE scheme with key privacy (informally defined as given a ciphertext, it is hard to guess the encryption public key) and weak wrong-key decryption property plus an anonymous commitment scheme. The encryption algorithm $ct \leftarrow \mathsf{OHE1.Enc}_{pk}(pt)$ for an arbitrary size plaintext pt combines (a) a commitment to the public key pk, along with (b) an FHE encryption of the plaintext pt, and c) an FHE encryption of the randomness r of the commitment: altogether this constitutes the ciphertext ct. Then the evaluator can mask a ciphertext ct for each user by the masking algorithm $\mathsf{OHE1.Mask}$ to obtain a ciphertext for each possible public key pk' such that it remains as valid encryption of m (i.e. it is decryptable to m) if the $pk' = pk$, and a harmless plaintext pt if $pk' \neq pk$. pt can be chosen by the evaluator freely based on the application. $\mathsf{OHE1.Mask}$ achieves this by homomorphically computing a commitment to pk' by first encrypting pk' with the FHE encryption algorithm, and then using the ciphertext of the randomness r. Then, it checks homomorphically whether the obtained commitment and the commitment given as part of ct are equal. Based on this comparison, the output of $\mathsf{OHE1.Mask}$ is a ciphertext decryptable to pt (if the check verifies) or a harmless plaintext e (if the check fails). The content of e can be chosen freely by the evaluator based on the application, e.g. if the subsequent homomorphically evaluated circuit is addition it can be 0. As the commitment scheme is assumed to be binding, the check can only verify for at most one public key, which completes the *isolation* property. We also provide optimizations in case the algorithms are repeatedly used by computing some parts of the algorithms only once.

OHE2. The second OHE scheme OHE2 has the same idea but the components of the encryption $ct \leftarrow \mathsf{OHE2.Enc}_{pk}(pt)$ are generated with a WWKE scheme with weak wrong-key decryption and key privacy instead of an FHE scheme with these properties. Then they are transfered to FHE ciphertexts by $\mathsf{OHE2.Mask}$, which then proceeds the same way as $\mathsf{OHE1.Mask}$. The main idea is to relax the FHE scheme with key privacy and weak wrong-key decryption in OHE1, as two separate encryption schemes in OHE2: (1) a standard FHE scheme and (2) a WWKE scheme.

Applications. Four applications of our OHE scheme are shown in this paper: a multi-client cloud service, an oblivious message retrieval scheme, a combination of the previous two for optimized bandwidth, as well as a fully private account-based cryptocurrency scheme. (1) The first scheme (for cloud computing) is obtained by including a digital signature in the encrypted query as proof of authorization. The signature is also encrypted for the anonymity of the query. The server homomorphically masks each query for each user first, homomorphically checks the signature, and processes the encrypted database depending on the signature check. Additionally, a NIZK scheme is used only for ensuring the correctness of the keys of clients. (2) The second application, oblivious message retrieval (OMR), only uses OHE as the building block: For each user, the message server obtains a t-bounded set of encrypted messages addressed to the user as follows. The server first masks all stored ciphertexts with $\mathsf{OHE.Mask}$. It then proceeds by homomorphically evaluating a circuit that filters for the respective messages, i.e. adding into a bucket the first message found that has not been masked or has not already been included. (3) The third application combines the previous two, to obtain an optimization in the multi-client cloud service, where each client only obtains her results of queries. Here OMR contributes to oblivious filtering of the query results in a given time period. (4) Finally, we show how a safe and fully private account-based cryptocurrency scheme results from plugging our OHE technique into the PriFHEte scheme [MS23] to obtain isolation, in addition to already achieved replay protection and transaction legitimacy by PriFHEte.

2 Related Work

In this section we relate our findings to the state of the art which we review with a special focus on the privacy of FHE applications.

Scheme	Key privacy	Isolation	Anonymous Multi-Client Cloud	Obliv. Msg. Retrieval	Anonymous Account-Based Cryptocurrency
Standard FHE	×	×	×	×	×
OMR1/2/3 [LT22]	✓	∗	×	✓	×
PriFHEte [MS23]	✓	∗∗	×	×	✓
Our OHE	✓	✓	✓	✓	✓

Table 1: Comparison of OHE with standard FHE, the OMR1, OMR2, and OMR3 schemes of [LT22], and the fully private account-based cryptocurrency scheme of [MS23], in terms of key privacy and isolation features, achieving privacy in multi-client cloud setting, OMR, and fully private transactions in account-based cryptocurrencies. ✓denotes the existence of a property or applicability of the given scheme to a setting. × denotes the negative of ✓. ∗ denotes the fact that OMR1 of [LT22] does not achieve isolation (as DoS attack is possible), and their OMR2 and OMR3 only achieves isolation for OMR applications. ∗∗ denotes the fact that [MS23] achieves isolation only for honestly generated transactions, and as we show in this work, double-spending is an imminent threat by maliciously generated transactions.

2.1 Fully homomorphic encryption (FHE)

FHE schemes [Gen09, SV10, DGHV10, BV11, BGV12, FV12, LATV12, Bra12, GSW13, BV14, DM15, CGGI16] enable to evaluate any boolean circuit on private input bits and receive the result in encrypted form. So far, the main focus of FHE research has been on improving efficiency. FHE by itself does not provide privacy of the key used for a ciphertext and the evaluated algorithm. Although the latter can be solved via evaluation of a universal circuit [KS08, SS09, KS16, ZYZL19], prior to this work, the former is not a securely solved problem.

2.2 Anonymous Encryption

Anonymous encryption techniques [BBDP01, LPQ12, BM20, HLH$^+$22] are enhanced encryption schemes for achieving *key privacy*, i.e. a property ensuring anonymity of the encryption key of a given ciphertext. They are useful in scenarios where a ciphertext may belong to multiple parties, such as anonymous communication through shared media. Although they are well-studied for communication in cryptographic literature, to the best of our knowledge, FHE schemes so far have not been proven or designed to be key-private. Further, even if some FHE schemes would be found to be key-private, the party that evaluates circuits on ciphertexts still needs to know the public key that encrypts a message as key privacy does not itself imply oblivious evaluation. The ElGamal encryption, which has already been shown to be key-private under decisional Diffie-Hellman assumption, has partial homomorphism and is applied on the privacy-preserving account-based transactions [BAZB19]. However, being only partially homomorphic, ElGamal is limited in its use for the evaluation of arbitrary circuits.

2.3 Secure Outsourced Computation

Privacy of outsourced computation (in server-client model) is crucial in but is not limited to outsourced private databases [AKSX04, LSP15, BEE$^+$17, BHE$^+$18, EZ19, BKK$^+$21, WBNM22], searchable encryption [CJJ$^+$13, CJJ$^+$14, KT19, DPPS20], outsourced cryptographic computations [HL05, LJLC12, CLT14, QYL$^+$22], machine learning and data mining [WCH$^+$07, NWI$^+$13, CXLC14, LC10], image processing [BZCP14, HWW$^+$16, WWH$^+$16], and outsourcing biometric recognition data [AL05, BA10, BA12, WHR$^+$15]. There has been extensive research in all of these mentioned areas for improving the efficiency of computations and privacy of the data content in different trust models, but unfortunately, the anonymity of the party requesting the task has not been a concern previously. To the best of our knowledge, even the above-mentioned generic FHE schemes do not achieve this anonymity, not to mention that they do not achieve isolation.

2.4 Anonymous Messaging

Although privacy of delivered message content has been well studied and applied, many messaging services available in practice fall short of hiding the metadata of the messages. Yet, in recent years, anonymous messaging (AM) has been a center of attention [WCGFJ12, CGBM15, Lun18]. The main challenge has been obliviously filtering the messages to a particular sender (by a message delivery server) and handing only the messages pertinent to a user. [LT22] provides three separate oblivious message retrieval protocols as solutions to the AM problem: The first one, "OMR1", uses a key-private FHE scheme but is prone to DoS attacks; The two others ("OMR2" and "OMR3") are not prone to this attack but contain specific optimizations of OMR1 for anonymous messaging (hence are not applicable to evaluation of arbitrary circuits) and their security is based only on LWE assumption.

2.5 Anonymity in Cryptocurrencies

Privacy of the financial actors has been considered of prime importance in cryptography applications. Pseudonyms (e.g. public keys) are already found to be vulnerable to simple tracing (as in the case of Bitcoin [Nak08]), and 'mix-in" type of constructions are at the risk of traceability attacks [KFTS17, MSH$^+$18] (as in the case with Monero [mon]). Zerocoin [MGGR13] and Zerocash [SCG$^+$14] achieve anonymity in transaction-based cryptocurrencies. Yet, transaction-based cryptocurrencies have constraints due to the scattering of each participant's deposit wherefore there is a growing trend towards account-based cryptocurrencies such as Ethereum [Eth], Filecoin [Fil], Ripple [Rip], and GOC-ledger [Lav23]. Some privacy-preserving payment schemes QuisQuis [FMMO19], Zether [BAZB19] and anonymous Zether [Dia21] have been proposed for account-based cryptocurrencies. Although they achieve confidentiality of the transferred amount, the privacy of the involved parties is not completely established, i.e. at best they achieve k-anonymity among a larger set of users. In contrast to these schemes, our solution provides full anonymity of both the sender and the receiver among the set of *all users*, not a subset of them. [MS23] claims to achieve the same property but as we show in this paper, their scheme is vulnerable to double-spending by increasing other users' balances more than the amount deducted.

Table 1 provides an overview of what OHE achieves w.r.t. standard FHE, [MS23], and [LT22], which target similar settings to our OHE scheme.

3 Preliminaries

This section first provides the notation used throughout this work, and then proceeds with essential background for our constructions in Section 5, namely: FHE schemes, wrong-key encryption (WKE) schemes, commitment schemes, and simple constructions of some boolean circuits. We then provide the basics of NIZK schemes, digital signature schemes, pseudorandom functions (PRF) which are used in our constructions in Section 6.

3.1 Notation

Throughout this paper,

- λ denotes the security parameter,
- $a \leftarrow B$ denotes a is picked from the set B at uniformly random,
- $a \leftarrow \mathsf{C}_k(z)$ denotes a is assigned as the output of the algorithm C executed on input z and the key k (we use this inside schemes),
- $\mathsf{C}_k(z) \leftarrow a$ denotes a is assigned as the output of the algorithm C executed on input z and the key k (we use this in definitions),
- $A \approx_c B$ denotes A is computationally indistinguishable from B,
- $\mathcal{A}(z) \rightarrow B$ denotes that the adversary $\mathcal{A}$ generates B on the input z.

3.2 Fully Homomorphic Encryption (FHE)

This work follows the fully homomorphic encryption (FHE) notation of the previous works [BV11, BGV12, MS23] with only a slight modification for simplicity (by abstracting out bit encryptions as encryptions of arbitrary strings). Formally, an FHE scheme consists of the polynomial time algorithms ($\mathsf{FHE.Setup}, \mathsf{FHE.KeyGen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Recrypt}, \mathsf{FHE.Eval}$), which are defined as follows:

$\mathsf{FHE.Setup}(1^\lambda) \to params$: The setup algorithm takes as input a unary security parameter 1^λ and outputs the system parameters $params$ that is used in the key generation (e.g. the group).

$\mathsf{FHE.KeyGen}(params, d) \to (pk, evk, sk)$: The key generation algorithm here deterministically generates the keys on randomness $d \in D$. It takes as input $params$ and a randomness d, and outputs the FHE public, evaluation, and secret keys (pk, evk, sk). pk and evk are public and are sometimes together referred to as the public key in the previous works, but we separate them for clarity. sk is kept as the secret key as in the conventional public key encryption schemes.

$\mathsf{FHE.Enc}_{pk}(pt) \to ct$: The encryption algorithm takes as input an FHE public key pk and an arbitrary length plaintext pt. It outputs the ciphertext ct. We acknowledge that the FHE schemes usually only encryptions of one-bit plaintexts, in which case, the de facto computation of the ciphertext ct is $ct_i \leftarrow \mathsf{FHE.Enc}_{pk}(pt_i)$ for each bit pt_i of the plaintext pt and then $ct := (ct_1, \ldots, ct_{|pt|})$. Nevertheless, we use this bulk encryption notation throughout the paper for convenience.

$\mathsf{FHE.Dec}_{sk}(ct) \to pt$: The decryption algorithm takes as input an FHE secret key sk and an arbitrary length ciphertext ct. It outputs the plaintext pt. Again, the actual computation usually takes place as first parsing ct for ciphertexts ct_i of plaintexts bits p_i and then decrypting for each bit $pt_i := \mathsf{FHE.Dec}_{sk}(ct_i)$ of pt. Also, we use this bulk decryption notation throughout the paper for convenience.

$\mathsf{FHE.Eval}(pk, C, CT) \to CT'$: The homomorphic evaluation algorithm takes as input an FHE public key pk, a circuit C compiled as a combination of XOR and AND operations, and a set CT of ciphertexts for the inputs to C. It outputs the set CT' of ciphertexts of the outputs from C.

$\mathsf{FHE.Recrypt}_{evk}(ct_{old}) \to ct_{new}$: The known FHE schemes employ a recrypt algorithm used within the evaluation operation to reduce the accumulated noise. Although it is usually considered as a sub-algorithm of $\mathsf{FHE.Eval}$ (and its use is not explicitly stated in this case), we also use it separately in our construction. The algorithm takes as input a noisy ciphertext ct_{old} and outputs a cleaner ciphertext ct_{new} by homomorphically decrypting ct_{old} such that $\mathsf{FHE.Dec}_{sk}(ct_{old}) = \mathsf{FHE.Dec}_{sk}(ct_{new})$.

The FHE scheme is assumed to satisfy CPA security [Gen09] briefly defined as follows:

Definition 1 (CPA security)**.** *A scheme* ($\mathsf{FHE.Setup}, \mathsf{FHE.KeyGen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Recrypt}, \mathsf{FHE.Eval}$) *has CPA security if:*

$$(pk, evk, \mathsf{FHE.Enc}_{pk}(0)) \approx_c (pk, evk, \mathsf{FHE.Enc}_{pk}(1))$$

where $(pk, evk, sk) \leftarrow \mathsf{FHE.KeyGen}(\mathsf{FHE.Setup}(1^\lambda))$.

We slightly alter the conventional full homomorphism [Gen09] so that for all honestly generated keys, full homomorphism would be satisfied. It is given as follows:

Definition 2 (full homomorphism)**.** *A scheme* ($\mathsf{FHE.Setup}, \mathsf{FHE.KeyGen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Recrypt}, \mathsf{FHE.Eval}$) *has full homomorphism if:*

1. *It is compact, that is, for any boolean circuit C with n bit output,* $\mathsf{FHE.Eval}(C, \cdots)$ *outputs a ciphertext bit string of size* $n \cdot poly(\lambda)$*,*
2. *It is homomorphic, that is, for any circuit C and any respective inputs* $pt_1, \ldots, pt_\ell$

$$\mathsf{FHE.Dec}_{sk}\Big(\mathsf{FHE.Eval}_{evk}\big(C, (ct_1, \ldots, ct_\ell)\big)\Big) = C(pt_1, \ldots, pt_\ell)$$

where $(pk, evk, sk) \leftarrow \mathsf{FHE.KeyGen}\big(\mathsf{FHE.Setup}(1^\lambda)\big)$*, and* $ct_i \leftarrow \mathsf{FHE.Enc}_{pk}(pt_i)$*.*

For simplicity, we prefer not to use the full homomorphism definition of [Gen09] that allows exceptions with negligible probability to the rule as this would require further consideration for maliciously generated keys and ciphertexts. Yet, we stress that for some standard FHE schemes, this can be achieved by converting the key generation algorithm to deterministic on a given outside randomness if the homomorphism always holds for a large and clearly definable subset D of generated keys. This can simply be obtained by FHE schemes [BV11, Bra12, BGV12, FV12, BLLN13], by restricting the noise distribution within a bound in the key generation as described by those works.

3.3 Key Privacy

Briefly, given a ciphertext and a set of public keys, *key privacy* is defined as a property (of an encryption scheme) that prevents the detection of which key has been used for generating the ciphertext [BBDP01]. In this work, we are more interested in CPA secure key-private schemes (not indistinguishability under the chosen ciphertext attack) since CPA security is the expected privacy guarantee from FHE schemes. Formally, key privacy is defined as follows:

Definition 3 (Key privacy)**.** *An encryption scheme* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *has key privacy iff for a bit* $b \in \{0, 1\}$

$$\big(b, pk_0, pk_1, \mathsf{Enc}_{pk_0}(b)\big) \approx_c \big(b, pk_0, pk_1, \mathsf{Enc}_{pk_1}(b)\big)$$

where $(pk_0, sk_0) \leftarrow \mathsf{KeyGen}(1^\lambda)$ *and* $(pk_1, sk_1) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$*.*

When we talk about key privacy of more specific schemes with some further public information, this information is always included in both sides of the computational indistinguishability. For example, in the case of the key privacy of the FHE schemes, (evk_0, evk_1) should also be included.

3.4 Wrong-Key Encryption (WKE)

We briefly describe the wrong-key encryption (WKE) scheme as formulated by [LT22] and used in the construction of [MS23]. It is a bitwise encryption scheme $(\mathsf{WKE.KeyGen}, \mathsf{WKE.Enc}, \mathsf{WKE.Dec})$ with key privacy and wrong-key decryption property in addition to classical CPA security.

$\mathsf{WKE.KeyGen}(1^\lambda) \to (pk, sk)$: Takes as input a unary security parameter 1^λ and outputs the WKE public-secret key pair (pk, sk).

$\mathsf{WKE.Enc}_{pk}(pt) \to ct$: Takes as input a WKE public key pk and an arbitrary length plaintext pt, and outputs the ciphertext ct. In fact, WKE definition of [MS23] enables only encryptions of one-bit plaintexts, hence this is a simplification of the notation similar to the one of $\mathsf{FHE.Dec}$ in Section 3.2.

$\mathsf{WKE.Dec}_{sk}(ct) \to pt$: Takes as input an WKE secret key sk and an arbitrary length ciphertext ct, and outputs the plaintext pt. Again, this is a simplification of the notation similar to the one of $\mathsf{FHE.Dec}$ in Section 3.2.

Apart from correctness (i.e. $\mathsf{WKE.Dec}_{sk}\big(\mathsf{WKE.Enc}_{pk}(pt)\big) = pt$), the WKE scheme is assumed to satisfy CPA security, key privacy [BBDP01], and wrong-key decryption [MS23]. The latter is defined as follows:

Definition 4 (Wrong-key decryption)**.** *A WKE scheme* $(\mathsf{WKE.KeyGen}, \mathsf{WKE.Enc}, \mathsf{WKE.Dec})$ *has wrong-key decryption if there exists a negligible function* $negl(\cdot)$ *s.t. the following inequality holds*

$$Pr\Big[\mathsf{WKE.Dec}_{sk_0}\big(\mathsf{WKE.Enc}_{pk_1}(b)\big) = b\Big] \leq 1/2 + negl(\lambda)$$

where $(pk_0, sk_0) \leftarrow \mathsf{WKE.KeyGen}(1^\lambda)$ *and* $(pk_1, sk_1) \leftarrow \mathsf{WKE.KeyGen}(1^\lambda)$ *and* $b \in \{0, 1\}$*.*

an FHE scheme can also be a wrong-key encryption (as in [LT22]) or it can be a separate encryption scheme (as in [MS23]).

3.5 (Anonymous) Commitment Schemes

A commitment scheme $(\mathsf{CSetup}, \mathsf{Commit}, \mathsf{COpen})$ is a generic cryptographic primitive that enables a party to commit to a value $v \in V$ (in general a bit or a bit string) for another party, for whom v stays hidden until the former opens it for the latter. The CSetup algorithm takes as input the security parameter 1^λ, and outputs the parameters *params* (e.g. the group). The commit algorithm computes $c := \mathsf{Commit}(params, v, r)$ where $r \in R$ is some randomness. c can be later opened by sending $\mathsf{COpen} = (v, r)$, which, in turn, can be checked for whether the equality $c{=}\mathsf{Commit}(params, v, r)$ holds. The two traditional requirements for commitment schemes are being *hiding* and *binding*, which are defined as follows:

Definition 5 (Computationally Hiding)**.** *A commitment scheme* $(\mathsf{CSetup}, \mathsf{Commit}, \mathsf{COpen})$ *is computationally hiding if for any* $v_0, v_1 \in V$

$$\big(params, v_0, v_1, \mathsf{Commit}(params, v_0, r_0)\big) \approx_c \big(params, v_0, v_1, \mathsf{Commit}(params, v_1, r_1)\big)$$

where $params \leftarrow \mathsf{Setup}(1^\lambda)$ *and* $r_0 r_1 \in R$ *picked randomly hidden from the distinguisher's view.*

Definition 6 (Computationally Binding)**.** *A commitment scheme* $(\mathsf{CSetup}, \mathsf{Commit}, \mathsf{COpen})$ *is computationally binding if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* $negl(\cdot)$ *such that*

$$Pr\big[\mathcal{A} \to (v_0, v_1, r_0, r_1) \in V^2 \times R^2 : \mathsf{Commit}(v_0, r_0) = \mathsf{Commit}(v_1, r_1)\big] \leq negl(\lambda)$$

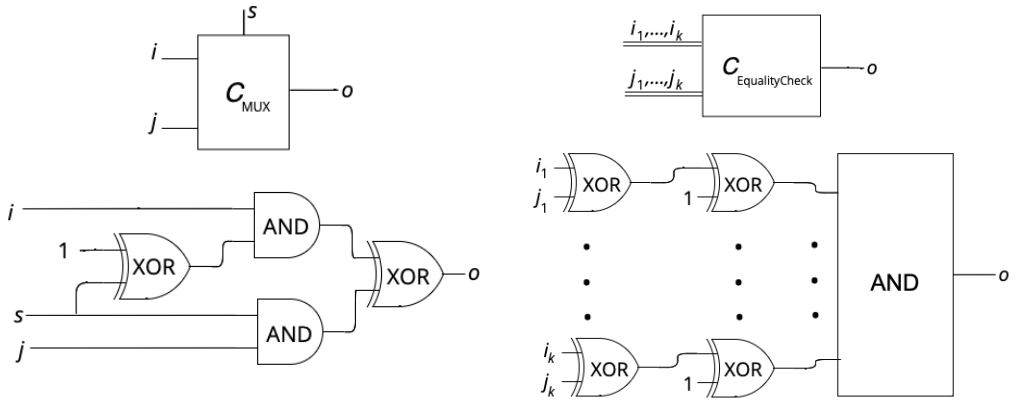
Apart from these, we want the commitment scheme to be *anonymous* (i.e. leaking no information about who has generated a given commitment c). Although the above traditional definition easily satisfy anonymity (if *params* is public and usable to generate commitments by any party), the universal commitment schemes [CF01, DG03, FLM11, Fuj14, Fuj16] are disqualified for black-box use as they intentionally leak the identity of the commitment's generator. Further, we need the Commit algorithm to be convertible to a boolean circuit C_{Commit}. Some of the commitment schemes in the literature that satisfy our criteria are [Ped92, DDN91, DCIO98, JKPT12, XXW13, CF13, Kim20]

3.6 Multiplexer and Equality Check Circuits

We provide brief descriptions of a multiplexer circuit C_{MUX} and an equality check circuit $C_{\mathsf{EqualityCheck}}$. The $C_{\mathsf{MUX}}(s, i, j)$ takes as input a selection bit s and two equal length bit strings i and j. It outputs i if $s = 0$. Otherwise, it outputs j. $C_{\mathsf{EqualityCheck}}(i, j)$ takes as input two equal length bit strings i and j. It outputs a bit, which is equal to 1 if $i = j$. Otherwise, it outputs 0. Figure 1 presents their constructions from XOR and AND gates that can be evaluated with $\mathsf{FHE.Eval}$.

3.7 Digital Signatures

A digital signature scheme $(\mathsf{SKeyGen}, \mathsf{Sign}, \mathsf{SVerify})$ is used for generating signatures on arbitrary length messages for proving authenticity. Its algorithms and assumed security property existential unforgeability are formally defined as follows:



(a) C_{MUX} for bit inputs i and j and one bit select s to obtain the output bit o

(b) $C_{\mathsf{EqualityCheck}}$ for k bit inputs $i = (i_1, \ldots, i_k)$ and $j = (j_1, \ldots, j_k)$ to obtain the output bit o

Figure 1: The circuit C_{MUX} for bit inputs i and j and one bit select s to obtain the output bit o is presented in (a) (the above one is the black-box, the below one is the construction). For arbitrary length k inputs, the circuit can just be evaluated on bits of the same index of both inputs. $C_{\mathsf{EqualityCheck}}$ for k bit inputs $i = (i_1, \ldots, i_k)$ and $j = (j_1, \ldots, j_k)$ to obtain the output bit o is presented in (b) (the above one is the black-box, the below one is the construction). The large AND gate can be evaluated with two input AND gates.

$\mathsf{SKeyGen}(1^\lambda) \to (vk, \sigma k)$: The key generation algorithm takes as input a unary security parameter 1λ, and outputs a verification key vk and a signing key σk.

$\mathsf{Sign}_{\sigma k}(m) \to \sigma$: The signing algorithm takes as input a signing key σk and an arbitrary length message m, outputs a signature σ.

$\mathsf{SVerify}_{vk}(m, \sigma) \to b$: The verification algorithm takes as input a verification key vk, an arbitrary length message m, and a signature σ. It outputs a bit b.

Definition 7 (Existential Unforgeability)**.** *A digital signature scheme (*$\mathsf{SKeyGen}, \mathsf{Sign}, \mathsf{SVerify}$ *is existentially unforgeable if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* $negl(\cdot)$ *such that*

$$Pr\left[\mathcal{A}(\mathcal{O}_{\mathsf{Sign}_{\sigma k}}) \to (m, \sigma) : \mathsf{SVerify}_{vk}(m, \sigma) = 1\right] \leq negl(\lambda)$$

where $\mathcal{O}_{\mathsf{Sign}_{\sigma k}}$ *is the signature oracle of the signing key* σk*,* $(vk, \sigma k) \leftarrow \mathsf{SKeyGen}(1^\lambda)$*, and* m *is not queried to the oracle before.*

3.8 Non-Interactive Zero-Knowledge Proofs (NIZK)

Non-interactive zero knowledge proofs of knowledge (NIZK) are used for non-interactively proving knowledge of a secret that satisfy a non-secret NP language. They can be obtained via the Fiat-Shamir transformation [FS87] of zero knowledge proof of knowledge protocols or with zk-SNARKs [Gro10, BSCTV14, PHGR16] (more efficiently) or with zk-STARKs [BSBHR18, MT21] (without trusted setup). We use the following notation. $\pi \leftarrow \mathsf{NIZK}(A, \mathsf{R}, B)$ denotes a NIZK of the private value set A such that the public value set B is in the relation R. NIZK schemes also have efficient verification algorithms $\mathsf{NVerify}(\pi, \mathsf{R}, B)$ that returns 1 if the proof π proves $B \in \mathsf{R}$ and returns 0 otherwise. They are assumed to securely realize the following ideal functionality:

Definition 8 (NIZK functionality)**.** *NIZK functionality is defined as an ideal functionally that acts based on the queries as follows:*

- *If queried with* $\mathsf{NIZK}(A, \mathsf{R}, B)$*: If* $B \in \mathsf{R}$*, return back a randomly picked proof* π*, record* (B, R, π)*. Otherwise, return a null value* $\perp$*.*
- *If queried with* $\mathsf{NVerify}(\pi, \mathsf{R}, B)$*: If the inputs are recorded before return 1. Otherwise, return 0.*

We highlight that in OHE we use the NIZK scheme only for consistency of public keys but not for honest generation of them. Picked randomnesses within the key generation algorithm is treated as a secret to be proven, but whether these randomnesses are picked in truly random is not proven as it is not necessary.

3.9 Pseudorandom Functions (PRF)

Pseudorandom functions are keyed functions that are assumed to emulate random functions. Formally, a PRF consists of two algorithms $\mathsf{PRF.Keygen}$ and PRF defined as follows:

$\mathsf{PRF.Keygen}(1^\lambda) \to k$: The PRF key generation algorithm takes as input a unary security parameter 1^λ and outputs a PRF key k.

$\mathsf{PRF}_k(m) \to p$: The PRF algorithm takes as input a PRF key k and an input string $m \in i_{\mathsf{PRF}}$ and outputs a pseudorandom value $p \in o_{\mathsf{PRF}}$. The domains i_{PRF} and o_{PRF} of inputs and outputs can be defined based on the application.

To define the security of PRF, we need to define also a random function RF.

$\mathsf{RF}(m) \to p$: The random function takes as input an input string $m \in i_{\mathsf{PRF}}$ and outputs a value $p \in o_{\mathsf{PRF}}$. If m is queried for the first time, p is picked from the domain uniformly at random. For other queries, the same output p is returned for the same input m. The domains i_{PRF} and o_{PRF} are the same as those of PRF.

Definition 9 (Pseudorandom function (PRF))**.** *A scheme* $(\mathsf{PRF.Keygen}, \mathsf{PRF})$ *is a PRF if for all adversary* $\mathcal{A}$ *there exists a negligible function* $negl(\cdot)$ *such that*

$$|Pr[A^{\mathsf{PRF}_k} \to 1] - Pr[A^{\mathsf{RF}} \to 1]| \leq negl(\lambda)$$

where $k \leftarrow \mathsf{PRF.Keygen}(1^\lambda)$ *and kept secret,* A^{PRF_k} *denotes* $\mathcal{A}$ *has access to oracle for* PRF_k*,* A^{RF} *denotes* $\mathcal{A}$ *has access to an oracle for* RF*.*

4 Why Isolation is Important: Double-Spending on PriFHEte

The PriFHEte cryptocurrency system proposed by Madathil and Scafuro in [MS23] is of special interest to us as it suggests that one can achieve full privacy for account-based cryptocurrencies with a WKE scheme and an FHE scheme. We first provide an overview of their approach and then lay out an attack that exploits the fact that their scheme has no strict isolation of transactions. In the subsequent Section 5, we will then introduce our solution to this isolation problem.

4.1 Overview of PriFHEte

The PriFHEte scheme essentially uses a WKE scheme to encrypt the transactions and an FHE scheme to encrypt each user's balance. For each user, the ledger manager (e.g. the miners) holds the WKE public key, the FHE public and evaluation keys, and a ciphertext for the WKE secret key encrypted under her FHE public key. The user's balance stays on the public ledger in ciphertext form under the user's FHE public key (i.e. Alice's balance v stays as a ciphertext c such that $v = \mathsf{FHE.Dec}_{sk_{Alice}}(c)$). Then, to send the amount u to Bob, Alice generates a transaction encrypting u both with her and Bob's WKE public keys and generating the related NIZK for the legitimacy of the transaction and some countermeasure proof against replay attack. Receiving a transaction the miners just process all accounts as both a possible sender and a possible receiver. Upon checking the proofs, for each possible sender

using the ciphertext for the WKE secret key encrypted under her FHE public key, they translate the ciphertext for the amount u being sent as an encryption under that user's WKE public key. Then, they provide an algorithm to be run by the miners to, supposedly, homomorphically isolate the ciphertext (i.e. to make it 0 if it was not initially encrypted under the WKE public key of the user). Then, the obtained ciphertext is evaluated by FHE for deducting the underlying plaintext from the user's amount. For each possible receiver, the miner also proceeds similarly, differing only at the final stage by adding the amount homomorphically. We briefly provide algorithms in their scheme as follows:

Create Account. To create an account, a user P_i generates

1. $(\mathsf{WKE}.pk_i, \mathsf{WKE}.sk_i) \leftarrow \mathsf{WKE.KeyGen}(1^\lambda)$,
2. $(\mathsf{FHE}.pk_i, \mathsf{FHE}.evk_i, \mathsf{FHE}.sk_i) \leftarrow \mathsf{FHE.KeyGen}(params)$,
3. $k - ct_i \leftarrow \mathsf{FHE.Enc}_{\mathsf{FHE}.pk_i}(WKE.sk_i)$,
4. $\mathsf{PK}_i = (\mathsf{WKE}.pk_i, \mathsf{FHE}.pk_i, k - ct_i)$.

P_i then publishes PK_i on the ledger. Also, the balance v of P_i rests in the ledger as an FHE ciphertext c_i (s.t. $v \leftarrow \mathsf{FHE.Dec}_{\mathsf{FHE}.sk_i}(c_i)$).

Private Payment. To send the amount v to a recipient with PK_r, the sender with PK_s runs the following algorithms:

1. $\mathsf{C}_s \leftarrow \mathsf{WKE.Enc}_{\mathsf{WKE}.pk_s}, (\mathsf{WKE}.pk_s)$,
2. $\mathsf{C}_r \leftarrow \mathsf{WKE.Enc}_{WKE.pk_r}(\mathsf{WKE}.pk_r)$,
3. $\mathsf{C}_d \leftarrow \mathsf{WKE.Enc}_{\mathsf{WKE}.pk_s}(-v)$,
4. $\mathsf{C}_c \leftarrow \mathsf{WKE.Enc}_{\mathsf{WKE}.pk_r}(v)$, and
5. Some proof π for legitimacy.

$\mathsf{PK_s}$ then sends the transaction $(\mathsf{C}_s, \mathsf{C}_r, \mathsf{C}_d, \mathsf{C}_c, \pi)$ to the ledger. The ledger writer then verifies the validity of π. If the verification fails the transaction is dropped. Otherwise, the ledger writer executes the following algorithms for each account PK_i to update the balance ciphertext c_i:

1. For being a potential sender:
   (a) $\mathsf{C}_{id} \leftarrow \mathsf{FHE.Eval}\Big(C_{\mathsf{WKE.Dec}}, \big(k - ct_i, \mathsf{FHE.Enc}_{\mathsf{FHE}.pk_i}(\mathsf{C}_s)\big)\Big)$
   (b) $\mathsf{C}_{amount} \leftarrow \mathsf{FHE.Eval}\Big(C_{\mathsf{WKE.Dec}}, \big(k - ct_i, \mathsf{FHE.Enc}_{\mathsf{FHE}.pk_i}(\mathsf{C}_d)\big)\Big)$
   (c) $\mathsf{C}_{flag} \leftarrow \mathsf{FHE.Eval}\Big(C_{\mathsf{EqualityCheck}}, \big(\mathsf{C}_{id}, \mathsf{FHE.Enc}_{\mathsf{FHE}.pk_i}(\mathsf{WKE}.pk_i)\big)\Big)$
   (d) $\mathsf{C}_v \leftarrow \mathsf{FHE.Eval}\big(C_{\mathsf{Multiplication}}, (\mathsf{C}_{flag}, \mathsf{C}_{amount})\big)$
   (e) $\mathsf{C}_i \leftarrow \mathsf{FHE.Eval}\big(C_{\mathsf{Addition}}, (\mathsf{C}_i, \mathsf{C}_v)\big)$
2. For being a potential receiver:
   (a) $\mathsf{C}_{id} \leftarrow \mathsf{FHE.Eval}\Big(C_{\mathsf{WKE.Dec}}, \big(k - ct_i, \mathsf{FHE.Enc}_{\mathsf{FHE}.pk_i}(\mathsf{C}_r)\big)\Big)$
   (b) $\mathsf{C}_{amount} \leftarrow \mathsf{FHE.Eval}\Big(C_{\mathsf{WKE.Dec}}, \big(k - ct_i, \mathsf{FHE.Enc}_{\mathsf{FHE}.pk_i}(\mathsf{C}_c)\big)\Big)$
   (c) $\mathsf{C}_{flag} \leftarrow \mathsf{FHE.Eval}\Big(C_{\mathsf{EqualityCheck}}, \big(\mathsf{C}_{id}, \mathsf{FHE.Enc}_{\mathsf{FHE}.pk_i}(\mathsf{WKE}.pk_i)\big)\Big)$
   (d) $\mathsf{C}_v \leftarrow \mathsf{FHE.Eval}\big(C_{\mathsf{Multiplication}}, (\mathsf{C}_{flag}, \mathsf{C}_{amount})\big)$
   (e) $\mathsf{C}_i \leftarrow \mathsf{FHE.Eval}\big(C_{\mathsf{Addition}}, (\mathsf{C}_i, \mathsf{C}_v)\big)$

Here, $C_{\mathsf{WKE.Dec}}$, $C_{\mathsf{Multiplication}}$, and C_{Addition} are the circuit representations of $\mathsf{WKE.Dec}$, conventional addition and multiplication operations, respectively. $C_{\mathsf{EqualityCheck}}$ abstracts out the operations described by the authors as first bitwise negating the plaintext of C_{id}, then XORing it with the $\mathsf{FHE.Enc}_{\mathsf{FHE}.pk_i}(\mathsf{WKE}.pk_i)$, and then multiplying each of the resulting bit ciphertexts. This is because our attack does not depend on its details.

4.2 Attack on PriFHEte

We describe a double-spending attack (similar to the DoS attack acknowledged by [LT22] on their OMR1 scheme) on the PriFHEte scheme where two parties PK_s and PK_e collude (we call them the "gang"). Whenever PK_s sends some amount v to a separate party PK_r, instead of executing the given algorithm as is, the gang compute C_r and C_c such that if they are decrypted with $\mathsf{WKE.sk}_e$, $\mathsf{WKE.pk}_e$ and v' would be obtained. The concrete algorithm of this attack is given in Algorithm 1.

Algorithm 1 Double-Spending Attack on PriFHEte

for each index i of a WKE public key **do**
  do
    $\mathsf{C}_{r,i} \leftarrow \mathsf{WKE.Enc}_{\mathsf{WKE}.pk_r}(\mathsf{WKE.pk}_{r,i})$
    $b \leftarrow \mathsf{WKE.Dec}_{\mathsf{WKE}.sk_e}(\mathsf{C}_{r,i})$
  while $b \neq \mathsf{WKE.pk}_{e,i}$

for each index i of an amount **do**
  do
    $\mathsf{C}_{c,i} \leftarrow \mathsf{WKE.Enc}_{\mathsf{WKE}.pk_r}(v_i)$
    $b \leftarrow \mathsf{WKE.Dec}_{\mathsf{WKE}.sk_e}(\mathsf{C}_{c,i})$
  while $b \neq v'_i$

$\mathsf{C}_r = (\mathsf{C}_{r,1}, \ldots, \mathsf{C}_{r,|\mathsf{WKE}.pk_r|})$
$\mathsf{C}_c = (\mathsf{C}_{c,1}, \ldots, \mathsf{C}_{c,|v_i|})$

PK_s computes the proof π of the transaction as given in the protocol description. The proof π can still be constructed honestly as C_r and C_c are indeed encryptions of $\mathsf{WKE}.pk_r$ and v under the public key $\mathsf{WKE}.pk_r$ (but now they are also encryptions of $\mathsf{WKE}.pk_e$ and v' under the public key $\mathsf{WKE}.pk_e$). Upon computation, PK_s sends the transaction to the miners. The transaction passes the checks for π. Then, the amount v is deducted from the PK_s's balance and added to PK_r's balance. However, PK_e also receives an additional amount of v'. Further improvements of this attack by preparing the ciphertexts to benefit more colluding parties are possible, but we omit their details.

Recall that the WKE scheme executes on bitwise plaintexts and the probability ϵ that the decrypted bit with the wrong key equals the encrypted one is less or equal to $1/2 + negl(\lambda)$, in which case it can be decrypted to 1 and 0 with probability ϵ and $1 - \epsilon$ based on the initially encrypted bit. The attack cost is an expected $O(|\mathsf{WKE}.pk_r| + |v|)$ number of WKE encryptions and decryptions for randomly picked $\mathsf{WKE}.pk_r$ and v. We highlight that the attack described here is possible for even honestly generated keys. Further attacks may be possible if the gang colludes in key generation.

4.3 Failed Fix Attempts

(In)applicability of NIZKs. A solution with including a NIZK proof of the generated ciphertext (such as knowledge of the randomness) would not be enough due to the following reasons. If the NIZK is given as a part of the ciphertext without being encrypted, the evaluator would see which public key the verification succeeds (as the verification results are visible), breaking the key privacy. If they are given as encrypted, then the attack idea would still be applicable as the generated ciphertext would be openable to multiple plaintexts due to wrong-key decryption.

Altering the underlying FHE or WKE Schemes. There could be two ways that we can think of to fix the attack in Section 4.2: (1) making the probability of success at each trial a negligible probability, (2) allowing WKE encryption to take as input plaintext bit strings and hoping for decryptions with different keys would not return a meaningful plaintext (e.g. v'). One way to achieve (1) would be ensuring wrong-key decryption would always return a fixed bit 0 or 1. Unfortunately, we do not know any encryption scheme that would satisfy this. It is even harder when the gang can generate their keys by colluding, which is a quite possible scenario in cryptocurrency applications.

Attempts for (2) fail if there exist sufficiently large number of meaningful plaintexts (e.g. v' could have a value from a large set of amounts). Even if the plaintext is a strict value (e.g. a public key), this is hard to achieve when the gang can generate their keys by colluding. We exemplify this by letting the WKE scheme to be the ElGamal encryption ($\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}$). It has both wrong-key decryption and key privacy. Given a discrete logarithm group $(q, g, \mathbb{G})$ of order q the ElGamal scheme briefly described as follows: KeyGen picks a random $s \in (0, \ldots, q)$, computes $h \leftarrow g^s$, and outputs $pk = h$ and $sk = s$. $\mathsf{Enc}_{pk}(m)$ picks a random $r \in (0, \ldots, q)$ outputs the ciphertext $c \leftarrow (g^r, mh^r)$. $\mathsf{Dec}_{sk}(c)$ parses (c_1, c_2) and outputs the plaintext $m \leftarrow c_2/c_1^s$.

Now to obtain a ciphertext c that can be decrypted to targeted messages $pk_0 = h_0$ and $pk_1 = h_1$ under two public keys sk_0 and sk_1, respectively (more concretely, to obtain the keys $sk_0 = s_0$, $pk_0 = h_0$, sk_1 and $pk_1 = h_1$ and the randomness r such that $(g^r, h_0 h_0^r) = (g^r, h_1 h_1^r),)$, one can run the following attack: Pick s_0, s_1, and r such that $s_0(r+1) = s_1(r+1) \mod q$ holds. This can easily be done by picking any two of those and then solving for the other. Then, $h_0 h_0^r = g_0^s (g_0^s)^r = g^{s_0(r+1)}$ and $h_1 h_1^r = g_1^s (g_1^s)^r = g^{s_1(r+1)}$ are equal.

Instead of working on a WKE scheme to obtain useful results, in this work, we take a different approach and even relax the wrong-key decryption requirement. In the next section, we provide our two OHE solutions. We provide a solution to the full privacy problem of account-based cryptocurrencies in Section 6.4 which is not vulnerable to double-spending.

5 Oblivious Homomorphic Encryption

In this section, we describe two novel OHE schemes in detail. We start by formally defining OHE, then we provide our construction obtained by black-box use of FHE and commitment schemes. We note that for generality, OHE is not an authenticated scheme by itself. Instead, achieving authenticity is left to the application. As we show in Section 6, this can be achieved by a digital signature (as in the cloud application), or a NIZK scheme (as in the account-based cryptocurrency application), or may not be needed (as in the OMR application).

Regarding the definition of OHE, the main difference from standard FHE schemes is the inclusion of a $\mathsf{OHE.Mask}$ algorithm. Formally, an OHE scheme consists of the polynomial time algorithms ($\mathsf{OHE.Setup}, \mathsf{OHE.KeyGen}, \mathsf{OHE.Enc}, \mathsf{OHE.Dec}, \mathsf{OHE.Mask}, \mathsf{OHE.Eval}$), which are defined as follows:

$\mathsf{OHE.Setup}(1^\lambda) \rightarrow params$: The setup algorithm takes as input a unary security parameter 1^λ and outputs the system parameters $params$ (e.g. groups).

$\mathsf{OHE.KeyGen}(params, d) \rightarrow (pk, evk, mk, sk)$: Similar to $\mathsf{FHE.KeyGen}$, the key generation algorithm here deterministically generates the keys on randomness $d \in D$. The key generation algorithm takes as input $params$ and randomness d. It outputs the OHE public key pk, the evaluation key evk, the masking key mk, and the secret key sk. pk is given to the parties that are expected to send encrypted messages, and the evaluator, (evk, mk) are given to the evaluator, sk is kept as secret.

$\mathsf{OHE.Enc}_{pk}(pt) \rightarrow ct$: The encryption algorithm takes as input an OHE public key pk and an arbitrary length plaintext pt. It outputs the ciphertext ct.

$\mathsf{OHE.Dec}_{sk}(ct) \rightarrow pt$: The decryption algorithm takes as input an OHE secret key sk and an arbitrary length ciphertext ct. It outputs the plaintext pt.

$\mathsf{OHE.Mask}_{mk}(ct, e) \rightarrow ct'$: The masking algorithm is executed for neutralizing the ciphertexts for irrelevant users by the evaluator. It takes as input an OHE masking key mk, a ciphertext ct, and a neutral string e such that $|e| = |\mathsf{OHE.Dec}_{sk}(ct)|$. It outputs a new ciphertext ct' such that it is decryptable to the same value (i.e. $\mathsf{OHE.Dec}_{sk}(ct) = \mathsf{OHE.Dec}_{sk}(ct')$) if the keys pk and mk would match, otherwise it is decryptable to e with sk (i.e. $e =$

$\mathsf{OHE.Dec}_{sk}(ct')$).

$\mathsf{OHE.Eval}_{evk}(C_{\mathsf{Alg}}, CT) \rightarrow CT'$: The homomorphic evaluation algorithm takes as input an OHE evaluation key $\mathsf{OHE.evk}$, a circuit C_{Alg} compiled as a combination of XOR and AND operations to compute the polynomial time algorithm Alg, and a set CT of ciphertexts of the input bits to Alg. It outputs the set CT' of ciphertexts of the output bits from Alg.

$\mathsf{OHE.Mask}$ neutralizes any ciphertext ct for any public key other than the one used in its encryption by converting it into a ciphertext ct' of a given neutral plaintext e, while it does not alter the underlying plaintext for the correct public key. We do not specify a value for e as it is application-dependent. For example, if the OHE scheme is used for the privacy of account-based cryptocurrencies in a similar manner to [MS23] then e can be 0 as this value will be added to the receiver's account. As another example, if the OHE scheme is used for confidentiality of update queries on databases and the server is just evaluating a universal circuit [KS08, SS09, KS16, ZYZL19] on the query and the database e can be $(\mathsf{XOR}, 0, \ldots, 0)$ to convert the query into a harmless one for other databases.

OHE inherits CPA security and full homomorphism from FHE and key privacy from [BBDP01] as in [LT22], which are as follows:

Definition 10 (CPA security)**.** *An OHE scheme* ($\mathsf{OHE.Setup}, \mathsf{OHE.KeyGen}, \mathsf{OHE.KeyVerify}$, $\mathsf{OHE.Enc}, \mathsf{OHE.Dec}, \mathsf{OHE.Mask}$) *has CPA security if:*

$$\big(pt_0, pt_1, pk, evk, mk, \mathsf{OHE.Enc}_{pk}(pt_0)\big) \approx_c \big(pt_0, pt_1, pk, evk, mk, \mathsf{OHE.Enc}_{pk}(pt_1)\big)$$

for any pt_0 *and* pt_1 *such that* $|pt_0| = |pt_1|$ *where* $(pk, evk, mk, sk) \leftarrow \mathsf{OHE.KeyGen}(\mathsf{OHE.Setup}(1^\lambda))$.

Definition 11 (Key privacy)**.** *An OHE scheme* ($\mathsf{OHE.Setup}, \mathsf{OHE.KeyGen}, \mathsf{OHE.KeyVerify}$, $\mathsf{OHE.Enc}, \mathsf{OHE.Dec}, \mathsf{OHE.Mask}$) *has key privacy if for any* $pt \in \{0,1\}^{poly(\lambda)}$

$$\begin{aligned}\big(pt, pk_0, evk_0, mk_0, pk_1, evk_1, mk_1, &\mathsf{OHE.Enc}_{pk_0}(pt)\big) \approx_c \\ &\big(pt, pk_0, evk_0, mk_0, pk_1, evk_1, mk_1, \mathsf{OHE.Enc}_{pk_1}(pt)\big)\end{aligned}$$

where $params \leftarrow \mathsf{OHE.Setup}(1^\lambda)$, $(pk_0, evk_0, mk_0, sk_0) \leftarrow \mathsf{OHE.KeyGen}(params)$, *and* $(pk_1, evk_1, mk_1, sk_1) \leftarrow \mathsf{OHE.KeyGen}(params)$.

Full homomorphism slightly changes due to the introduction of $\mathsf{OHE.Mask}$ algorithm compared to that of FHE. Similar to the full homomorphism of FHE, that of OHE is also defined deterministically. This is useful in cases where ciphertexts can be maliciously generated. The formal definition is as follows:

Definition 12 (Full homomorphism)**.** *An OHE scheme* ($\mathsf{OHE.Setup}, \mathsf{OHE.KeyGen}$ $\mathsf{OHE.Enc}, \mathsf{OHE.Dec}, \mathsf{OHE.Mask}$) *has full homomorphism if:*

1. *It is compact, that is, for any boolean circuit* C *with* n *bit output,* $\mathsf{OHE.Eval}(C, \cdots)$ *outputs a ciphertext bit string of size* $n \cdot poly(\lambda)$,
2. *It is homomorphic, that is, for any circuit* C *and any respective input strings* $pt_1, \ldots, pt_\ell$

$$\mathsf{OHE.Dec}_{sk}\Big(\mathsf{OHE.Eval}_{evk}\big(C, (ct_1, \ldots, ct_\ell)\big)\Big) = C(pt_1, \ldots, pt_\ell)$$

   where $(pk, evk, mk, sk) \leftarrow \mathsf{OHE.KeyGen}\big(\mathsf{OHE.Setup}(1^\lambda)\big)$, $ct_i \leftarrow \mathsf{OHE.Enc}_{pk}(pt_i)$, *and* $|pt_i| = |e|$ *for* $i \in (1, \ldots, \ell)$,

3. *It is masked homomorphic, that is, for any circuit* C, *any respective input strings* $pt_1, \ldots, pt_\ell$, *and any string* e

$$\mathsf{OHE.Dec}_{sk}\Big(\mathsf{OHE.Eval}_{evk}\Big(C, \big(\mathsf{OHE.Mask}_{mk}(ct_1, e), \ldots, \mathsf{OHE.Mask}_{mk}(ct_\ell, e)\big)\Big)\Big) = C(pt_1, \ldots, pt_\ell)$$

   where $(pk, evk, mk, sk, \pi) \leftarrow \mathsf{OHE.KeyGen}\big(\mathsf{OHE.Setup}(1^\lambda)\big)$, $ct_i \leftarrow \mathsf{OHE.Enc}_{pk}(pt_i)$, *and* $|pt_i| = |e|$ *for* $i \in (1, \ldots, \ell)$.

We define the isolation property of the OHE scheme as follows:

Definition 13 (Isolation)**.** *An OHE scheme* ($\mathsf{OHE.Setup}, \mathsf{OHE.KeyGen}, \mathsf{OHE.KeyVerify}, \mathsf{OHE.Enc}, \mathsf{OHE.Dec}, \mathsf{OHE.Mask}$) *provides isolation if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* $negl(\cdot)$ *s.t. the following inequality holds:*

$$Pr\Big[\mathcal{A} \to (ct, e) \text{ s.t. } e \neq \mathsf{OHE.Dec}_{sk_0}\big(\mathsf{OHE.Mask}_{mk_0}(ct, e)\big), \\ e \neq \mathsf{OHE.Dec}_{sk_1}\big(\mathsf{OHE.Mask}_{mk_1}(ct, e)\big)\Big] \leq negl(\lambda)$$

given that $|e| = |\mathsf{OHE.Dec}_{sk_0}(ct)| = |\mathsf{OHE.Dec}_{sk_1}(ct)|$, $(pk_0, evk_0, mk_0, sk_0) \leftarrow \mathsf{OHE.KeyGen}(params)$*, and* $(pk_1, evk_1, mk_1, sk_1) \leftarrow \mathsf{OHE.KeyGen}(params)$

The isolation property allows the adversary to generate the ciphertext ct and the neutral string e for keys generated by a challenger. This definition is sufficient in scenarios where the users have incentives to not collude for generating their keys maliciously, e.g. OMR applications (as making a DoS attack by sending a message valid for oneself and another party does not make much sense). In applications where key generation may also take place maliciously, e.g. cryptocurrencies, honest generation can be ensured with NIZK proofs.

5.1 Construction: OHE1

We have already seen FHE and WKE-based schemes [MS23, LT22] fail themselves to obtain OHE evident from the described attack in Section 4.2. Appending a NIZK proof (or its encrypted form) does not solve the issue due to the reasons mentioned in Section 4.3. Even in our OHE schemes if the commitment included in a ciphertext was given in encrypted form (instead of the evaluator executing the encryption), then such an attack would be applicable. We indeed need a scheme that binds a public key to a ciphertext, but also hides the public key. Thus the use of a commitment to the public key may make sense.

We now describe the first OHE construction OHE1 in detail. The scheme uses, in a black-box manner, an FHE scheme with key privacy and weak wrong-key decryption, and an anonymous commitment scheme. This is not the first work to make this assumption on FHE schemes as [LT22] assumes an FHE scheme with key privacy and even wrong-key decryption described in Section 3.4. Our weak wrong-key decryption property is even a relaxation of the wrong-key decryption property and is easier to obtain and prove, by only requiring decryption under a different key to return a bit. It is formally defined as follows:

Definition 14 (Weak wrong-key decryption)**.** *An encryption scheme (*$\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}$*) has weak wrong-key decryption if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* $negl(\cdot)$ *s.t. the following inequality holds*

$$Pr\Big[\mathcal{A} \to (pk, sk', b) \text{ s.t. } b \in \{0,1\}, b' \leftarrow \mathsf{Dec}_{sk'}\big(\mathsf{Enc}_{pk}(b)\big), b' \notin \{0,1\}\Big] \leq negl(\lambda),$$

where pk *is a valid public key and* sk' *is a valid secret key (i.e. they are in the output space of* KeyGen*).*

Here, the key privacy and CPA security properties ensure the decryptions under a different key do not reveal information about the public key pk and the plaintext, respectively. The adversary can freely pick pk and sk', they are not necessarily a key pair. We provide our OHE1 scheme in Algorithm 2.

Let us explain what we have done in Algorithm 2. $\mathsf{OHE1.KeyGen}$ generates the keys with $\mathsf{FHE.KeyGen}$ and sets $mk = (pk, evk)$[1]. $\mathsf{OHE1.Enc}$ algorithm generates a ciphertext ct with three parts (ct_1, ct_2, ct_3). ct_3 provides a commitment to the public key of the encrypted message. As the commitment scheme is hiding and anonymous, it does not reveal any information about who the ciphertext belongs. The main use of it here is that it privately binds the ciphertext to pk. ct_2 just provides a ciphertext for the randomness r used in the commitment, which is used by $\mathsf{OHE1.Mask}$ for homomorphically generating a ciphertext of the commitment later.

[1]So this scheme does not require the additional burden of a separate masking key.

Algorithm 2 OHE1

procedure $\mathsf{OHE1.Setup}(1^\lambda)$
Compute $params_f \leftarrow \mathsf{FHE.Setup}(1^\lambda)$ and $params_c \leftarrow \mathsf{CSetup}(1^\lambda)$
Output $params = (params_f, params_c)$

procedure $\mathsf{OHE1.KeyGen}(params, d)$
Compute $(pk, evk, sk) \leftarrow \mathsf{FHE.KeyGen}(params_f, d)$
Set $mk = (pk, evk)$
Output (pk, evk, mk, sk)

procedure $\mathsf{OHE1.Enc}_{pk}(pt)$
Pick $r \leftarrow R$
Compute $ct_1 \leftarrow \mathsf{FHE.Enc}_{pk}(pt)$
Compute $ct_2 \leftarrow \mathsf{FHE.Enc}_{pk}(r)$
Compute $ct_3 \leftarrow Commit(params_c, pk, r)$ ▷ Appending a commitment to pk
output $ct = (ct_1, ct_2, ct_3)$

procedure $\mathsf{OHE1.Dec}_{sk}(ct)$
Parse $(ct_1, \ldots) = ct$
Compute and output $pt \leftarrow \mathsf{FHE.Dec}_{sk}(ct_1)$

procedure $\mathsf{OHE1.Mask}_{mk}(ct, e)$ ▷ Homomorphically checks the commitment
Parse $(ct_1, ct_2, ct_3) = ct$
Compute:
$\hat{ct_1} \leftarrow \mathsf{FHE.Recrypt}_{evk}(ct_1)$
$\hat{ct_2} \leftarrow \mathsf{FHE.Recrypt}_{evk}(ct_2)$
$\hat{pk} \leftarrow \mathsf{FHE.Enc}_{pk}(pk)$
$\hat{e} \leftarrow \mathsf{FHE.Enc}_{pk}(e)$
$\hat{ct_3} \leftarrow \mathsf{FHE.Enc}_{pk}(ct_3)$
$\hat{params_c} \leftarrow \mathsf{FHE.Enc}_{pk}(params_c)$
$\hat{ct} \leftarrow \mathsf{FHE.Eval}_{evk}\big(C_{\mathsf{Commit}}, (\hat{params_c}, \hat{pk}, \hat{ct_2})\big)$
$\hat{b} \leftarrow \mathsf{FHE.Eval}_{evk}\big(C_{\mathsf{EqualityCheck}}, (\hat{ct}, \hat{ct_3})\big)$
$ct' \leftarrow \mathsf{FHE.Eval}_{evk}\big(C_{\mathsf{MUX}}, (\hat{b}, \hat{e}, \hat{ct_1})\big)$ ▷ Decides the plaintext of the final ciphertext
Output ct'

procedure $\mathsf{OHE1.Eval}_{evk}(C, CT)$
for each ciphertext $ct_i \in CT$ **do**
Parse $(ct_{i,1}, \ldots) = ct_i$
Append $ct_{i,1}$ to T
Compute and output $CT' \leftarrow \mathsf{FHE.Eval}_{evk}(C, T)$

OHE1.Mask first recrypts with FHE.Recrypt both ct_1 and ct_2 (hence obtaining $\hat{ct_1}$ and $\hat{ct_2}$), as a countermeasure to the case that the algorithm is running with a different public key (similar to [LT22]). We know that the ciphertexts are still decryptable as the FHE scheme has the weak wrong-key decryption property, but we do not know how noisy these ciphertexts are under that key. So, direct use of it by FHE.Eval could have the risk of ambiguous and unexpected results. OHE1.Mask then generates ciphertexts for $\hat{pk}$, $\hat{e}$, $\hat{ct_3}$, and $\hat{params_c}$ as they are required for the subsequent evaluations. Having obtained the ciphertexts for the inputs of the commitment, i.e. $\hat{params_c}$, $\hat{pk}$, and $\hat{ct_2}$, OHE1.Mask homomorphically generates the ciphertext $\hat{ct}$ to the commitment. Then, OHE1.Mask homomorphically checks whether the plaintext commitment ct_3 of the ciphertext $\hat{ct_3}$ is equal to the underlying plaintext of the ciphertext $\hat{ct}$ by evaluating $C_{\mathsf{EqualityCheck}}$. OHE1.Mask obtains a ciphertext $\hat{b}$, which can only be an encryption of 1 in case pk is the correct key. Otherwise, it should be 0. Then using this ciphertext OHE1.Mask homomorphically selects which of the underlying plaintexts of $\hat{e}$ and $\hat{ct_1}$ goes to the output by evaluating C_{MUX}.

Simple Optimization. If the masking algorithm is applied repeatedly by the evaluator for a given user with the same e (which we expect to be the general case), the evaluator may generate $\hat{pk}$, $\hat{e}$, and $\hat{params_c}$ once, and may use them repeatedly. This does not cause any security breach, as their values are not hidden, and they are encrypted for the sake of FHE evaluation on them.

Security. We provide the security theorem below and its proof (sketch) in Appendix A.1:

Theorem 1. *If:*

- *the underlying FHE scheme has CPA security, full homomorphism, weak wrong-key decryption, and key privacy, and*
- *the underlying (anonymous) commitment scheme is computationally hiding and computationally binding;*

then the OHE1 scheme has CPA security, key privacy, full homomorphism, and isolation.

5.2 Construction: OHE2

We now describe the second OHE construction OHE2 which is an application (similar to [MS23], see Section 4.1 for details) of a WWKE (weak WKE) scheme (WWKE.Enc, WWKE.KeyGen, WWKE.Dec) to OHE1. Here, we just use any FHE scheme instead of the one with key privacy and weak wrong-key encryptions as in OHE1. Yet, we require another encryption scheme WWKE with key privacy and weak wrong-key decryption (in addition to having a circuit $C_{\mathsf{WWKE.Dec}}$ for decryption). WWKE scheme can be instantiated with learning with error (LWE) based scheme as in [MS23]. Similar to [MS23], WWKE is used for encryption, then the ciphertext is transformed to FHE ciphertexts for further evaluation. Yet, unlike [MS23], our scheme benefits from the binding property of the anonymous commitment scheme as in OHE1 to achieve isolation.

The main differences of this scheme from OHE1 are as follows. OHE2.KeyGen generates additionally the keys for the WWKE scheme. Notably, a transfer key tk is generated as an FHE ciphertext of the WWKE secret key. OHE2.Enc algorithm just replaces FHE encryptions with WWKE encryptions by first encrypting them with FHE and then decrypting the underlying WWKE ciphertext by using the circuit $C_{\mathsf{WWKE.Dec}}$ for the WWKE.Dec algorithm. This circuit depends on the WWKE scheme used. OHE2.Dec just anticipates an incoming ciphertext maybe a freshly generated WWKE ciphertext or an FHE ciphertext obtained after a circuit evaluation and decrypts accordingly. OHE2.Mask computes $\hat{ct_1}$ and $\hat{ct_2}$ as homomorphically decrypting the initial WWKE ciphertexts by using the key tk. It then proceeds as in OHE1.Mask.

Simple Optimization. Similar to OHE1, If the masking algorithm is be applied repeatedly by the evaluator for a given user with the same e (which we expect to be the general case), the evaluator may generate $\hat{\mathsf{FHE}.pk}$, $\hat{e}$, and $\hat{params_c}$ once, and may use them repeatedly.

Security. We provide the security theorem below and its proof (sketch) in Appendix A.2:

Theorem 2. *If:*

- *the underlying WWKE scheme has CPA security, weak wrong-key decryption, and key privacy,*

Algorithm 3 OHE2

procedure $\mathsf{OHE2.Setup}(1^\lambda)$
Compute $params_f \leftarrow \mathsf{FHE.Setup}(1^\lambda)$ and $params_c \leftarrow \mathsf{CSetup}(1^\lambda)$
Output $params = (params_f, params_c)$

procedure $\mathsf{OHE2.KeyGen}(params, d)$ ▷ The keys for both FHE and WWKE generated.
Compute $(\mathsf{FHE}.pk, \mathsf{FHE}.evk, \mathsf{FHE}.sk) \leftarrow \mathsf{FHE.KeyGen}(params_f, d)$
Compute $tk \leftarrow \mathsf{FHE.Enc}_{\mathsf{FHE}.pk}(\mathsf{WWKE}.sk)$
▷ Used for transfering the ciphertext to FHE by the evaluator.
Set $pk = (\mathsf{FHE}.pk, \mathsf{WWKE}.pk)$, $evk = \mathsf{FHE}.evk$, $mk = (\mathsf{FHE}.pk, \mathsf{FHE}.evk, tk)$
Set $sk = (\mathsf{FHE}.sk, \mathsf{WWKE}.sk)$
Output (pk, evk, mk, sk)

procedure $\mathsf{OHE2.Enc}_{pk}(pt)$
Pick $r \leftarrow R$, Compute $ct_1 \leftarrow \mathsf{WWKE.Enc}_{\mathsf{WWKE}.pk}(pt)$, $ct_2 \leftarrow \mathsf{WWKE.Enc}_{\mathsf{WWKE}.pk}(r)$
Compute $ct_3 \leftarrow Commit(params_c, pk, r)$, Output $ct := (ct_1, ct_2, ct_3)$

procedure $\mathsf{OHE2.Dec}_{sk}(ct)$
Parse $(ct_1, \ldots) = ct$
if ct is WWKE ciphertext **then**
Compute and output $pt \leftarrow \mathsf{WWKE.Dec}_{\mathsf{WWKE}.sk}(ct_1)$
else if ct is an FHE ciphertext **then**
Compute and output $pt \leftarrow \mathsf{FHE.Dec}_{\mathsf{FHE}.sk}(ct_1)$

procedure $\mathsf{OHE2.Mask}_{mk}(ct, e)$
Parse $(ct_1, ct_2, ct_3) = ct$
Compute:
$ct'_1 \leftarrow \mathsf{FHE.Enc}_{\mathsf{FHE}.pk}(ct_1)$, $ct'_2 \leftarrow \mathsf{FHE.Enc}_{\mathsf{FHE}.pk}(ct_2)$

$\hat{ct}_1 \leftarrow \mathsf{FHE.Eval}_{\mathsf{FHE}.evk}\big(C_{\mathsf{WWKE.Dec}}, (tk, ct'_1)\big)$
$\hat{ct}_2 \leftarrow \mathsf{FHE.Eval}_{\mathsf{FHE}.evk}\big(C_{\mathsf{WWKE.Dec}}, (tk, ct'_2)\big)$
▷ Transfering the WWKE ciphertexts to FHE by homomorphically decrypting them

$\hat{\mathsf{FHE}}.pk \leftarrow \mathsf{FHE.Enc}_{\mathsf{FHE}.pk}(\mathsf{FHE}.pk)$
$\hat{e} \leftarrow \mathsf{FHE.Enc}_{\mathsf{FHE}.pk}(e)$, $\hat{ct}_3 \leftarrow \mathsf{FHE.Enc}_{\mathsf{FHE}.pk}(ct_3)$
$\hat{params}_c \leftarrow \mathsf{FHE.Enc}_{\mathsf{FHE}.pk}(params_c)$
$\hat{ct} \leftarrow \mathsf{FHE.Eval}_{\mathsf{FHE}.evk}\big(C_{\mathsf{Commit}}, (\hat{params}_c, \hat{\mathsf{FHE}}.pk, \hat{ct}_2)\big)$
$\hat{b} \leftarrow \mathsf{FHE.Eval}_{\mathsf{FHE}.evk}(C_{\mathsf{EqualityCheck}}, (\hat{ct}, \hat{ct}_3))$
$ct' \leftarrow \mathsf{FHE.Eval}_{\mathsf{FHE}.evk}\big(C_{\mathsf{MUX}}, (\hat{b}, \hat{e}, \hat{ct}_1)\big)$
Output ct'

procedure $\mathsf{OHE2.Eval}_{evk}(C, CT)$
for each ciphertext $ct_i \in CT$ **do** Parse $(ct_{i,1}, \ldots) = ct_i$, Append $ct_{i,1}$ to $\bar{CT}$
Compute and output $CT' \leftarrow \mathsf{FHE.Eval}_{\mathsf{FHE}.evk}(C, \bar{CT})$

- *the underlying FHE scheme has CPA security and full homomorphism, and*
- *the underlying (anonymous) commitment scheme is computationally hiding and computationally binding;*

then OHE2 scheme has CPA security, key privacy, full homomorphism, and isolation.

6 Applications of OHE

In this section we show four black-box applications of OHE: a privacy-preserving multi-client cloud computing service, an oblivious message retrieval scheme, the combination of the previous two applications (for efficient downloading of results without jeopardizing privacy), and finally a fully anonymous account-based cryptocurrency.

6.1 Multi-Client Cloud Scheme

The classic outsourcing system consists of a multiple clients requesting computation service from a central provider. Each client uploads their data once and later requests a series of actions on it e.g. extract from or update their database content. The goal is to keep the content, the requests, and the results private, which classic FHE achieves. However, the service provider fully knows to which database a request applies, and then also which is the client-specific result. Besides adding privacy to the service metadata we wish to also prevent selectively denying, or slowing down requests for specific clients. We will now show how the service provider can be made oblivious to the identity of the clients, their content, and their results.

6.1.1 System Model

The system consists of a server (honest-but-curious) and n number of clients, potentially colluding with each other. Each client i has a database $\mathcal{DB}_i$ encrypted and then uploaded to the server. The channel between the server and the clients is assumed to be shared (i.e. each client's packets arrive to the server at the same port), anonymous (e.g. via WLAN or TOR network [TOR]), and sequential (so the packets do not interfere with each other). Ideally, each client can query only her own database (including database update queries). Also, ideally, the server handles the query by updating the database of the query owner and making the output available to the client. Further, the server is expected to handle the query without being able to learn any information about whose database it is updating. We only assume that the server handles a query correctly. We point to the existing literature for preventing incorrect execution e.g. verifiable computation [GGP10], probabilistically checkable proofs [TRMP12, GKR15] or incentivized outsourced computation [Kup17, KSN20, BYK21]. Clients are naturally incentivized to misbehave if, for example, it would be possible to alter another client's database, individually or by colluding: Clients may want, if possible, to collude so as to extract more database changes per request that the service plan permits by conjoining their queries. Also, some "clients" (e.g. competitors, or state-sponsored agents) may want to benefit from the service's privacy features and bring down a service by flooding it with requests for random public keys.

6.1.2 Construction

Equipped with OHE schemes, a solution to the above problems only requires the addition of a NIZK scheme for honest key generation and a proof of authorization for which a classic digital signature $(\mathsf{SKeyGen}, \mathsf{Sign}, \mathsf{SVerify})$ scheme suffices. We highlight that instead of OHE, an FHE scheme with key privacy and wrong-key encryption combined with a digital signature in a similar manner is insufficient, as two clients can still conjoin their queries. We provide the scheme in Algorithm 4.

The Setup algorithm is run in the beginning by the cloud server. The ClientJoin algorithm is run by any client that can dynamically join the system, and it generates the OHE keys, their proof of consistency, digital signature keys, and the encrypted database. The generated public keys are checked with the KeyVerify algorithm by the server and dropped if the algorithm fails. Otherwise, the keys

Algorithm 4 Privacy-Preserving Multi-Client Cloud Scheme

procedure SETUP(1^λ)
  $params \leftarrow \mathsf{OHE.Setup}(1^\lambda)$
  If any NIZK setup algorithm exists execute it here
procedure CLIENTJOIN($\mathcal{DB}$)
  Randomly pick $d \leftarrow D$
  $(pk, evk, mk, sk) \leftarrow \mathsf{OHE.KeyGen}(params, d)$
  Compute $\pi \leftarrow \mathsf{NIZK}\big(d, \mathsf{KR}, (pk, evk, mk)\big)$
  $(vk, \sigma k) \leftarrow \mathsf{SKeyGen}(1^\lambda)$
  $\hat{\mathcal{DB}} \leftarrow \mathsf{OHE.Enc}_{pk}(\mathcal{DB})$
  Send $(pk, evk, mk, \pi, vk, \hat{\mathcal{DB}})$ to the server, keep $(sk, \sigma k)$
procedure KEYVERIFY($pk, evk, mk, \pi, vk, \hat{\mathcal{DB}}$) ▷ Ensures the correctness of the keys
  if $\mathsf{NVerify}\big(\pi, \mathsf{KR}, (pk, evk, mk)\big)$ **then**
    Append (pk, evk, mk, vk, π) to PK
    Save the database $DB \leftarrow \mathsf{OHE.Mask}_{mk}(\hat{\mathcal{DB}})$
    Output 1 and return
  else
    Output 0 and return
procedure MAKEQUERY($q, pk, \sigma k$)
  $\hat{q} \leftarrow \mathsf{OHE.Enc}_{pk}(q)$
  $\hat{\sigma} \leftarrow \mathsf{OHE.Enc}_{pk}\big(\mathsf{Sign}_{\sigma k}(q)\big)$
  Send $Q = (\hat{q}, \hat{\sigma})$
procedure PROCESSQUERY($\hat{q}, \hat{\sigma}, pk, evk, mk, vk, DB_i$)
  ▷ Isolation of the ciphertext, also checks the signature for authenticity
  $\hat{mq} \leftarrow \mathsf{OHE.Mask}_{mk_i}\big(\hat{q}, (\text{Null Update, Output } \perp)\big)$
  $\hat{vk_i} \leftarrow \mathsf{OHE.Enc}_{pk_i}(vk_i)$
  $\hat{b} \leftarrow \mathsf{OHE.Eval}_{evk_i}\big(C_{\mathsf{SVerify}}, (\hat{vk_i}, \hat{mq}, \hat{m\sigma})\big)$
  $\hat{vq} \leftarrow \mathsf{OHE.Eval}_{evk_i}\big(Gate_{\mathsf{AND}}, (\hat{mq}, \hat{b})\big)$
  $(DB_i, output_i) \leftarrow \mathsf{OHE.Eval}_{evk_i}\big(C_{\mathsf{QueryProcess}}, (\hat{vq}, DB_i)\big)$
  Send $output_i$
relation KR
  $(pk, evk, mk) \in \mathsf{KR}$ if they are generated by $\mathsf{OHE.KeyGen}$ on $(params, d)$ and $d \in D$

are appended to the key list PK. Executed by a client, the MAKEQUERY algorithm takes as input a plaintext (possibly update) query q on the database, signs it with the client's secret key σk, and then encrypts both the query and signature with OHE to obtain the encrypted query $Q = (\hat{q}, \hat{\sigma})$.

Upon obtaining a query Q, the server executes the PROCESSQUERY algorithm for all client databases, ideally in parallel. The algorithm then neutralizes the query ciphertext $\hat{q}$ with $\mathsf{OHE.Mask}$ by turning the underlying plaintext as a null update (e.g. XOR database with 0s if the query is processed with the evaluation of a universal circuit as $C_{\mathsf{QueryProcess}}$) and some query output $\perp$. In fact, this application requires the evaluation of a circuit $C_{\mathsf{QueryProcess}}$ with identity queries. Yet, any circuit C with inputs a database $x = (x_1, \ldots, x_m)$ and a query $y = (y_1, \ldots, y_n)$ without an identity query can always be converted into a circuit C' with the same inputs and an additional input query bit y_{n+1} as in Figure 2. Then, for null update, it suffices to set $y_{n+1} = 1$, given that other queries set it as 0.

We note that such a neutralization is not needed for $\hat{\sigma}$ as the query itself became harmless for other databases. The algorithm then verifies the encrypted signature $\hat{\sigma}$ by homomorphically evaluating a circuit C_{SVerify} for $\mathsf{SVerify}$. If verification succeeds (i.e. the query maker is an authorized party), the

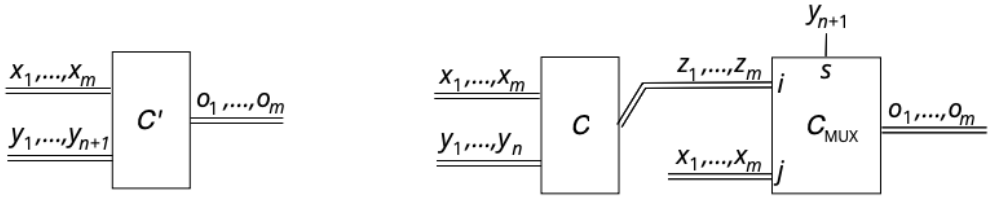


Figure 2: The circuit C' for inputs of a database $x = (x_1, \ldots, x_m)$ and a query $y = (y_1, \ldots, y_n, y_{n+1})$ (the left one is the black-box, the right one is the construction). If $y_{n+1} = 0$, the database remains unchanged. Otherwise, it outputs the updated database as $C(x, y)$.

subsequent process of the query with the database results in two things: an updated database and an output. Otherwise, the database remains unchanged (XORed with 0s) and a null output $\perp$ is obtained.

6.2 Oblivious Message Retrieval

In this section, we propose an oblivious message retrieval scheme (similar to OMR1 of [LT22]) that achieves anonymity of messages that are sent via a single server: each user is able to selectively download the messages for which they are the recipient.

6.2.1 System Model

The message server only knows the public keys (including pk, evk, and mk in the OHE application) of each user. In each time period, the server can obliviously filter encrypted messages that can be sent to any of the users to obtain a set of messages (the upper bound) that are pertinent to each user. t is the upper bound for the number of messages that can be sent through the system to each user in a time period. Then, the user only downloads these t messages, improving bandwidth efficiency. In this scheme, proof of authenticity is not needed as the sender of a message would be fully anonymous. Unlike OMR1 [LT22], which is based on the direct application of FHE, our scheme uses the OHE scheme and is not susceptible to the DoS attack observed by [LT22] where one can generate a message pertinent to multiple users[2].

6.2.2 Construction

Our construction is given in Algorithm 5. n messages, each of which has a plaintext length ℓ, are processed by the server for each user i and a set of t messages is obtained which are then sent to the user. The Setup algorithm is run by the server to produce the OHE parameters, which are then published. Each client runs the KeyGen algorithm to produce the OHE keys, gives the keys pk, evk, and mk to the server, and publishes pk. We highlight that $\mathsf{OHE.KeyVerify}$ and proof is not needed as the users do not have any incentive for colluding and weak isolation is enough. The generated keys are appended to the key list PK. Anyone who would like to send a message m to the client can encrypt the message as $c \leftarrow \mathsf{OHE.Enc}_{pk}(m)$ and send it to the server. For each time period, the server collects the message ciphertexts in a database; at the end of the time period, it runs Retrieve for each user to filter the messages that only are pertinent to them.

The Retrieve algorithm first runs over all the messages with $\mathsf{OHE.Mask}$ to obtain the flag Flag_i which is a ciphertext for the plaintext bit 1 for the pertinent messages. The other messages in the database are updated to the ciphertext of 1^ℓ, and their flags are set as a ciphertext of the bit 0. Then, the algorithm proceeds for each output message by setting it as a ciphertext for the first found pertinent message. It only replaces that message's underlying plaintext as non-pertinent so that it would not

[2]It may be argued that even if the DoS attack is prevented, spamming would be a way to deny a user from getting her legitimate messages. However, this can be disincentivized by charging for each message sent, implementable via an anonymous cryptocurrency.

Algorithm 5 Oblivious Message Retrieval Scheme

procedure SETUP(1^λ)
  $params \leftarrow \mathsf{OHE.Setup}(1^\lambda)$
procedure KEYGEN($params$)
  Randomly pick $d \leftarrow D$
  $(pk, evk, mk, sk) \leftarrow \mathsf{OHE.KeyGen}(params)$
  Send (pk, evk, mk) to the server. Keep sk.
procedure RETRIEVE(pk, evk, mk, M)
  for each message ciphertext m_i **do** ▷ Homomorphically isolate the pertinent messages.
    $\hat{m_i} \leftarrow \mathsf{OHE.Mask}_{mk}(m_i, 1^\ell)$
    $\mathsf{FlagInverse}_i \leftarrow \mathsf{OHE.Eval}_{evk}(C_{\mathsf{AND}}, \hat{m_i})$
    $\mathsf{Flag}_i \leftarrow \mathsf{OHE.Eval}_{evk}\Big(Gate_{\mathsf{XOR}}, \big(\mathsf{FlagInverse}_i, \mathsf{OHE.Enc}_{pk}(1)\big)\Big)$
  for each output index $j \in (1, \ldots, t)$ **do** ▷ Obtaining the t output messages
    Initialize as $o_j \leftarrow \mathsf{OHE.Enc}_{pk}(1^\ell)$, $\mathsf{OutputFlag}_j \leftarrow \mathsf{OHE.Enc}_{pk}(1)$
    for each message ciphertext $\hat{m_i}$ **do**
      $\hat{s_{ij}} \leftarrow \mathsf{OHE.Eval}_{evk}(Gate_{\mathsf{AND}}, \big(\mathsf{Flag}_i, \mathsf{OutputFlag}_j\big))$
      $o_j || \mathsf{OutputFlag}_j \leftarrow \mathsf{OHE.Eval}_{evk}\Big(C_{\mathsf{MUX}}, \big(\hat{s_{ij}}, o_j || \mathsf{OutputFlag}_j, \hat{m_i} || \mathsf{OHE.Enc}_{pk}(0)\big)\Big)$
        ▷ the output is set as the first pertinent message found
      $\hat{update} \leftarrow \mathsf{OHE.Eval}_{evk}\big(C_{\mathsf{EqualityCheck}}, (\hat{m_i}, o_j)\big)$
      $\hat{m_i} || \mathsf{Flag}_i \leftarrow \mathsf{OHE.Eval}_{evk}\Big(C_{\mathsf{MUX}}, \big(\hat{update}, \hat{m_i} || \mathsf{Flag}_i, \mathsf{OHE.Enc}_{pk}(1^\ell) || \mathsf{OHE.Enc}_{pk}(0)\big)\Big)$
        ▷ the message found is replaced with the ciphertext for a non-pertinent message
  Send all $OUT = (o_1, \ldots, o_t)$ to the user
procedure DECODE($sk, OUT = (o_1, \ldots, o_t)$)
  for each message ciphertext o_j **do**
    $m_j \leftarrow \mathsf{OHE.Dec}_{sk}(o_j)$
    Drop if $m_j = 1^\ell$

re-appear in the next output ciphertext. Instead, the next pertinent ciphertext will go to the next output.

6.3 Oblivious Retrieval of Query Result

Algorithm 4 for the oblivious cloud service in 6.1 has the drawback that the server must make all generated $output_i$ available to all clients because it does not know to whom a result belongs. By combining the computing service with the Oblivious Message Retrieval technique (Section 6.2, Algorithm 5), we can avoid this inefficiency, i.e. the client does not need to download all the results up on making a query. Instead, the server will now process the queries in batches with fixed time slots, which enables to batch download the results specific to a client. This improves the bandwidth efficiency for clients as they retrieve only their query results instead of downloading all batch output generated by the server. The system model is similar to the one in Section 6.1.1 with the main difference that a shared communication channel from the server to the client is not necessary.

6.3.1 Construction

Algorithm 6 presents our *Private Database with Oblivious Query Result Retrieval* scheme. Compared to Algorithm 4, the main difference is that PROCESSQUERY sets the output as a specific ciphertext 1^ℓ for non-pertinent results $\perp$. Then, at the end of the time period, the server filters the results for each

Algorithm 6 Oblivious Query Result Retrieval Scheme

procedure SETUP(1^λ)
  $params \leftarrow \mathsf{OHE.Setup}(1^\lambda)$
  If any NIZK setup algorithm exists execute it here

procedure CLIENTJOIN($\mathcal{DB}$)
  Randomly pick $d \leftarrow D$, compute $(pk, evk, mk, sk, \pi) \leftarrow \mathsf{OHE.KeyGen}(params, d)$
  $\pi \leftarrow \mathsf{NIZK}\big(d, \mathsf{KR}, (pk, evk, mk)\big)$, $(vk, \sigma k) \leftarrow \mathsf{SKeyGen}(1^\lambda)$, $\hat{\mathcal{DB}} \leftarrow \mathsf{OHE.Enc}_{pk}(\mathcal{DB})$
  Send $(pk, evk, mk, \pi, vk, \hat{\mathcal{DB}})$ to the server, keep $(sk, \sigma k)$

procedure KEYVERIFY($pk, evk, mk, \pi, vk, \hat{\mathcal{DB}}$) ▷ Ensures the correctness of the keys
  if $\mathsf{NVerify}\big(\pi, \mathsf{KR}, (pk, evk, mk)\big)$ **then**
    Append (pk, evk, mk, vk, π) to PK
    Save the database $DB \leftarrow \mathsf{OHE.Mask}_{mk}(\hat{\mathcal{DB}})$
    Output 1 and return
  else
    Output 0 and return

procedure MAKEQUERY($q, pk, \sigma k$)
  $\hat{q} \leftarrow \mathsf{OHE.Enc}_{pk}(q)$, $\hat{\sigma} \leftarrow \mathsf{OHE.Enc}_{pk}\big(\mathsf{Sign}_{\sigma k}(q)\big)$
  Send $Q = (\hat{q}, \hat{\sigma})$

procedure PROCESSQUERY($Q = (\hat{q}, \hat{\sigma}), pk, evk, mk, vk, DB_i$)
  $\hat{mq} \leftarrow \mathsf{OHE.Mask}_{mk_i}\big(\hat{q}, (\text{Null Update, Output } 1^\ell)\big)$
  $\hat{vk_i} \leftarrow \mathsf{OHE.Enc}_{pk_i}(vk_i)$
  $\hat{b} \leftarrow \mathsf{OHE.Eval}_{evk_i}\big(C_{\mathsf{SVerify}}, (\hat{vk_i}, \hat{mq}, \hat{m\sigma})\big)$
  $\hat{vq} \leftarrow \mathsf{OHE.Eval}_{evk_i}\big(Gate_{\mathsf{AND}}, (\hat{mq}, \hat{b})\big)$
  $(DB_i, output_i) \leftarrow \mathsf{OHE.Eval}_{evk_i}\big(C_{\mathsf{QueryProcess}}, (\hat{vq}, \hat{\mathcal{DB}}_i)\big)$
  Append $output$ to O

procedure RESULTRETRIEVE(pk, evk, mk, O)
  for each query output $output_i \in O$ **do**
    ▷ Masking is not needed as it is obtained during PROCESSQUERY.
    $\mathsf{FlagInverse}_i \leftarrow \mathsf{OHE.Eval}_{evk}(C_{\mathsf{AND}}, output_i)$
    $\mathsf{Flag}_i \leftarrow \mathsf{OHE.Eval}_{evk}\Big(Gate_{\mathsf{XOR}}, \big(\mathsf{FlagInverse}_i, \mathsf{OHE.Enc}_{pk}(1)\big)\Big)$
  for each output index $j \in (1, \ldots, t)$ **do** ▷ Obtaining the t output messages
    Initialize as $o_j \leftarrow \mathsf{OHE.Enc}_{pk}(1^\ell)$, $\mathsf{OutputFlag}_j \leftarrow \mathsf{OHE.Enc}_{pk}(1)$
    for each $output_i \in O$ **do**
      $\hat{s_{ij}} \leftarrow \mathsf{OHE.Eval}_{evk}\big(Gate_{\mathsf{AND}}, (\mathsf{Flag}_i, \mathsf{OuputFlag}_j)\big)$
      $o_j || \mathsf{OuputFlag}_j \leftarrow \mathsf{OHE.Eval}_{evk}\Big(C_{\mathsf{MUX}}, \big(\hat{s_{ij}}, o_j || \mathsf{OutputFlag}_j, output_i || \mathsf{OHE.Enc}_{pk}(0)\big)\Big)$
        ▷ the output is set as the first pertinent message found
      $\hat{update} \leftarrow \mathsf{OHE.Eval}_{evk}\big(C_{\mathsf{EqualityCheck}}, (output_i, o_j)\big)$
      $output_i || \mathsf{Flag}_i \leftarrow \mathsf{OHE.Eval}_{evk}\Big(C_{\mathsf{MUX}}, \big(\hat{update}, output_i || \mathsf{Flag}_i, \mathsf{OHE.Enc}_{pk}(1^\ell) || \mathsf{OHE.Enc}_{pk}(0)\big)\Big)$
        ▷ that result is replaced with the ciphertext for a non-pertinent result
  Send all $OUT = (o_1, \ldots, o_t)$ to the user

procedure DECODE($sk, OUT = (o_1, \ldots, o_t)$)
  for each message ciphertext o_j **do**
    $m_j \leftarrow \mathsf{OHE.Dec}_{sk}(o_j)$
    Drop if $m_j = 1^\ell$

relation KR
  $(pk, evk, mk) \in \mathsf{KR}$ if they are generated by $\mathsf{OHE.KeyGen}$ on $(params, d)$ and $d \in D$

user by ResultRetrieve similar to Retrieve procedure of 5, but without executing OHE.Mask as the ciphertexts are already masked during ProcessQuery.

6.4 Fully Privacy-Preserving Account-Based Cryptocurrency

In this section, we propose an account-based cryptocurrency scheme that achieves the privacy of the sender, the receiver, and the transferred amount. Our scheme is not susceptible to the attack that we described in Section 4.2.

6.4.1 System Model

The scheme is based on a public ledger whose contents can be read by any party on the Internet. The state of the ledger is based on eventual consistency among ledger managers (e.g. miners in proof-of-work based Bitcoin). A ledger manager updates its ledger replica based on a prescribed procedure that varies depending on the application. We want the users' balances to remain private on the ledger such that even the ledger managers do not know them. However, when a user (in this case called a sender) makes a transfer of some amount v to another user (in this case called a receiver), we want the ledger managers to obliviously update the balances of both parties. Although the ledger managers can act maliciously by e.g. running a different algorithm than the prescribed one or dropping transactions, we assume that they are honest-but-curious for practical purposes, similar to previously proposed blockchain applications (e.g. Zerocash [SCG+14] or e-donation [BK20]): The misbehavior of managers will be handled by the eventual consistency property of the ledger, which reflects reality (e.g. in the banking system) quite well.

6.4.2 Construction

The solution requires proof of legitimacy for the transaction, which we obtain by a NIZK scheme as in [MS23]. We also inherit the PRF-based replay prevention from [MS23]. Algorithm 7 provides the proposal: Transactions take place in separate time epochs (the one in which the transaction takes place is denoted ep) such that every party is allowed to make only one transaction at each epoch. The Setup algorithm takes place once at the beginning of cryptocurrency execution. If it includes private choices that may affect the security of the system, there may be decentralized solutions (such as the one of [BK19]).

The CreateAccount procedure is run by each user when they join the system to create an account. It basically generates the OHE keys, a PRF key and a commitment to it (for replay protection), and an unmasked account UnmAcc initialized as the balance 0 encrypted under the public key of the user (so only she can decrypt the balance). UnmAcc is appended to the list of accounts after masking as a. The account a is updated with every transaction taking place in the system, but the underlying plaintext value only changes with transactions relevant to the user. CreateAccount also generates a NIZK π_{Keys} and π_{Account} to ensure that initially generated keys are correct and the initial balance is 0, which is then checked by the ledger managers. If the checks by AccountCheck verifies, the ledger managers append the user data $(pk, evk, mk, c, a, \pi_{Keys}, \pi_{\mathsf{Account}})$ to the list of accounts A. It also holds the list $\mathsf{EP}_{\mathsf{PRF}}$ of used PRF outputs at each epoch ep.

The Spend procedure is executed by a sender to transfer the amount v to some receiver with an account on the ledger. It essentially generates two OHE ciphertexts ct_s and ct_r of the amounts v and $-v$ under the public keys pk_s and pk_r, respectively. Spend also generates a PRF output p generated from ep under the sender's PRF key k_s, hence if a user tries to make two transactions (or another party tries to replay the original transaction) in a given epoch, the PRF output check by the ledger managers will fail and the transaction will not be added. A NIZK π_{S} is also generated for the legitimacy of the transaction t. It essentially proves that t satisfies the transaction relation TR: TR shows that ct_s and ct_r are indeed valid encryptions to the same amount (ct_s being negative) under valid keys of account holders, generated by someone who knows the sender secret key sk_s, that the sender has enough balance on the ledger, and that the PRF output p is valid with the current epoch ep and the sender's PRF key k_s.

Algorithm 7 Account-Based Full Private Payment Scheme

procedure SETUP(1^λ)
$params_o \leftarrow \mathsf{OHE.Setup}(1^\lambda)$
$params_c \leftarrow \mathsf{CSetup}(1^\lambda)$
Execute NIZK setup algorithm (if any exists).
Output $params = (params_o, params_c)$ ▷ NIZK setup parameters may be appended

procedure CREATEACCOUNT($params_o, params_c$) ▷ Generates keys and an initial account
$(pk, evk, mk, sk) \leftarrow \mathsf{OHE.KeyGen}(params_o)$
$\pi_{\mathsf{Keys}} \leftarrow \mathsf{NIZK}\big(d, \mathsf{KR}, (pk, evk, mk)\big)$
$k_s \leftarrow \mathsf{PRF.KeyGen}(1^\lambda)$
$r \leftarrow R$, $c \leftarrow \mathsf{Commit}(params_c, k, r)$
$\mathsf{UnmAcc} = \mathsf{OHE.Enc}_{pk}(0)$
$\pi_{\mathsf{Account}} = \mathsf{NIZK}\big(\mathcal{R}, \mathsf{AR}, (pk, \mathsf{UnmAcc})\big)$ ▷ Proof of correctness of the account
Send $(pk, evk, mk, c, \mathsf{UnmAcc}, \pi_{\mathsf{Keys}}, \pi_{\mathsf{Account}})$ to the ledger managers. Keep sk, r, and k.

procedure ACCOUNTCHECK($pk, evk, mk, c, \mathsf{UnmAcc}, \pi_{\mathsf{Keys}}, \pi_{\mathsf{Account}}$)
if $\mathsf{NVerify}\big(\pi, \mathsf{KR}, (pk, evk, mk)\big) \& \mathsf{NVerify}\big(\pi_{\mathsf{Account}}, \mathsf{AR}, (pk, \mathsf{UnmAcc})\big) \& pk \notin A$ **then**
Append $(pk, evk, mk, c, a \leftarrow \mathsf{OHE.Mask}_{mk}(\mathsf{UnmAcc}), \pi_{\mathsf{Keys}}, \pi_{\mathsf{Account}})$ to A
Output 1 and return
else
Output 0 and return

procedure SPEND($A, \mathsf{EP_{PRF}}, pk_s, pk_r, v, sk_s, k_s, r_s, a_s, p, ep, c_s$)
$ct_s = \mathsf{OHE.Enc}_{pk_s}(-v)$ ▷ The value to be detucted
$ct_r = \mathsf{OHE.Enc}_{pk_r}(v)$ ▷ The value to be added
$p \leftarrow \mathsf{PRF}_{k_s}(ep)$ ▷ Protection against replay attack [MS23]
$\pi_{\mathsf{S}} = \mathsf{NIZK}\big((pk_s, pk_r, v, sk_s, k_s, r_s, a_s, R), \mathsf{TR}, (A, ep, ct_s, ct_r, p)\big)$
Send $\big(t = (ct_s, ct_r, p), \pi_{\mathsf{S}}\big)$ to the ledger managers

procedure PROCESSTRANSACTION($A, \mathsf{EP_{PRF}}, ep, ct_s, ct_r, p, \pi_{\mathsf{S}}$))
if $\mathsf{NVerify}\big(\pi_{\mathsf{S}}, \mathsf{TR}, (A, ep, ct_s, ct_r, p)\big) = 0$ or $p \in \mathsf{EP_{PRF}}$ **then** return
For being a potential sender:
$ct'_s \leftarrow \mathsf{OHE.Mask}_{mk_i}(ct_s, 0)$
$a_i \leftarrow \mathsf{OHE.Eval}_{evk_i}\big(C_{\mathsf{Addition}}, (a_i, ct'_s)\big)$
For being a potential receiver:
$ct'_r \leftarrow \mathsf{OHE.Mask}_{mk_i}(ct_r, 0)$
$a_i \leftarrow \mathsf{OHE.Eval}_{evk_i}\big(C_{\mathsf{Addition}}, (a_i, ct'_r)\big)$
Append p to $\mathsf{EP_{PRF}}$

relation KR
$(pk, evk, mk) \in \mathsf{KR}$ if they are generated by $\mathsf{OHE.KeyGen}$ on $(params, d)$ and $d \in D$
relation AR
$(pk, \mathsf{UnmAcc}) \in \mathsf{AR}$ if a is generated by $\mathsf{OHE.Enc}_pk(0)$ with randomness in $\mathcal{R}$
relation TR
$(A, ep, ct_s, ct_r, p) \in \mathsf{TR}$ if:
$ct_s = \mathsf{OHE.Enc}_{pk_s}(y)$ and $ct_r = \mathsf{OHE.Enc}_{pk_s}(z)$ with randomnesses in R and $y + z = 0$
sk_s is a valid secret key for pk_s
The value v is encrypted to obtain ct_r under the public key pk_r and randomnesses in R
v is smaller than the value obtained by decrypting $a_s \in A$ with secret key sk_s corresponding to pk_s
The same key k_s is committed at $c_s \in A$ (with randomness r_s) and is used at $p \in t$
$p = \mathsf{PRF}_{k_s}(ep)$

The ledger managers process the transaction t and the proof π_{S} for each user by applying ProcessTransaction on her account separately. ProcessTransaction first checks the proof π_{S} coming with the transaction t and whether p is used before. If the checks fail, it drops the transaction and the ledger managers terminate the further execution. Otherwise, ct_s and ct_r are first masked by $\mathsf{OHE.Mask}$ and then obliviously added to the user's balance.

7 Discussion and Future Work

In this section, we discuss our proposal's impact in terms of achieving a secure and practical approach and elaborate on future work. The discussion mainly focuses on efficiency, trustless computation, and potential improvements.

Efficiency. The first scheme, OHE1, is asymptotically as efficient as the underlying FHE scheme because the complexity of the additional operations only depends on the security parameter λ. In $\mathsf{OHE1.Enc}$, when compared to traditional FHE, we have only an additional FHE ciphertext ct_2 of a randomness r and a commitment ct_3 to the public key pk. Regarding the $\mathsf{OHE1.Mask}$ operation, the efficiency depends on the commitment scheme and specifically on the circuit compilation of C_{Commit}, but not on the encrypted message.

Regarding the second scheme OHE2, it is again asymptotically as efficient as the underlying WWKE and FHE scheme because the complexity of the additional operations only depends on the security parameter λ. Similar to $\mathsf{OHE1.Enc}$, $\mathsf{OHE2.Enc}$ has only an additional WWKE ciphertext ct_2 of a randomness r and a commitment ct_3 to the public key pk, in addition to ct_1, which is the WWKE ciphertext of the encrypted message. In fact, considering the state-of-the-art, the encryption algorithm of OHE2 can be more efficient than that of OHE1, as it can use a simple construction such as ElGamal. Also for the $\mathsf{OHE2.Mask}$ operation, the efficiency depends on the commitment scheme and specifically on the circuit compilation of $C_{\mathsf{WWKE.Dec}}$ and C_{Commit}, but not on the encrypted message.

Regarding the multi-client cloud application in Section 6.1, our solution is a direct application of the OHE scheme and a digital signature, so the efficiency depends on both schemes. They, however, do not depend on the query size and how it is processed which is only dependent on the underlying FHE scheme. Regarding our OMR scheme described in Section 6.2, the sent messages are just encrypted by $\mathsf{OHE.Enc}$, so their efficiency depends only on the underlying OHE scheme used. For filtering, the server iterates $O(n \cdot t)$ times over each message and calls the $\mathsf{OHE.Mask}$ algorithm where n is the number of messages in the database and t is the bound of messages a user may receive. This is comparable to the most efficient LWE-based OMR solution of [LT22] which requires filtering to be executed in $O\Big(N \cdot \big(\log(t \ + \epsilon_p \cdot N) \log(\epsilon_n^{-1}) \log^4(N) + \log(1/\epsilon_p)\big)\Big)$ time where ϵ_p and ϵ_n are the false positive and false negative rates, respectively (our solution does not have this issue). Regarding the fully private account-based cryptocurrency example in Section 6.4, transactions and their processing depend both on the OHE scheme and the NIZK scheme. The efficiency of the ledger manager is slightly worse than [MS23], as we require the computation of the commitment homomorphically.

We highlight that in this work we choose to remain as general as possible, therefore we provided multiple application examples. This is a reminiscence of the infamous efficiency vs. generality issue in secure computation research. Further improvements based on application-specific solutions (such as the OMR solutions of [LT22]) are probable and are left as future work.

Security of Underlying Primitives. OHE1 uses an FHE scheme with additional key privacy and weak wrong-key decryption properties. To the best of our knowledge, we still do not have an FHE scheme that provably achieves these properties. Yet, a weak wrong-key encryption scheme would be easier to obtain once a FHE scheme is proven to have key privacy. Regarding anonymous commitments, we already have plenty of provably secure solutions. All of these primitives may be further improved for additional security, e.g. to make them universally composable or secure against quantum computers.

OHE2 uses a WWKE scheme which is easy to obtain with LWE or RLWE-based encryption schemes such as Regev's [Reg05]. The other primitives that OHE2 uses are also standard FHE and NIZK, which are both hot research topics.

Trustless Computation. Our ultimate interest is in trustless (outsourced) computations. Yet, the

OHE scheme in this paper is designed under the assumption of an honest-but-curious evaluator. This gap can be closed by several existing solutions, for example hardening our approach by detecting malicious activity through verifiable computations [GGP10] or probabilistically checkable proofs [TRMP12, GKR15], or by incentivizing the evaluator to act honestly [Kup17, KSN20, BYK21]. We note that this hardening may be not needed for the distributed cryptocurrency application (Section 6.4) which has its own internal way of computing trust and reacting accordingly.

Regarding the OHE.Setup algorithms, the constructions do not necessitate a trusted setup. If the FHE and the commitment schemes do not require a trusted setup, nor do OHE schemes. Even if a scheme with a trusted setup is used, based on the application, this setup may be distributed (e.g. via a similar method to [BK19]).

8 Conclusion

For more than four decades, steady progress has been made towards code execution privacy: Privacy homomorphisms were first postulated in 1978 [RAD78]. From 1998 on, cryptographic homomorphic schemes were shown for restricted computation classes [ST98, SYY99]. Although the negative result of 2001 [BGI$^+$01] ruled out all classic obfuscation techniques for achieving privacy, it reasserted the case for general homomorphic encryption, should it exist. It was in 2009 when Gentry [Gen09] finally gave a construction and positive answer to this question. The quest is not over, though.

Our Oblivious Homomorphic Encryption scheme delivers on the wish for trustless computing: the identity of the code submitter remains hidden (key indistinguishability) and computation spaces remain separated (isolation property). Our construction is based on an anonymous commitment scheme used for individually masking the operations of the multiple clients: the executing host (being oblivious about the client identities) can only apply computation requests on the encrypted data of *all* clients in parallel, wherefore side effects must be computationally limited (masked) to each client's encrypted data.

Similar to all approaches above, our OHE scheme assumes that a semi-trusted execution platform exists from where the protected computation requests are launched and where results are received and inspected in plain. Although there exists generic solution in detecting malicious execution, ultimately, we would like to see "unstoppable computing" where services run in encrypted form in fully detached mode (without the need for a trusted home platform). Services would also control their own deployment to remote places in order to evade resource starvation attacks. Such a goal opens new operational and cryptographic challenges, for example, oblivious code migration, privacy-preserving request forwarding, and secure software updates.

Acknowledgements

We acknowledge support from the University of Basel, Switzerland. We thank Erick Lavoie and Ali Ajorian from the University of Basel for interesting discussions, comments, and reviews.

References

[AKSX04] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *ACM SIGMOD '04*, 2004.

[AL05] Mikhail J. Atallah and Jiangtao Li. Secure outsourcing of sequence comparisons. *IJIS*, 4(4), 2005.

[BA10] Marina Blanton and Mehrdad Aliasgari. Secure outsourcing of dna searching via finite automata. In *Data and Applications Security and Privacy XXIV*, 2010.

[BA12] Marina Blanton and Mehrdad Aliasgari. Secure outsourced computation of iris matching. *J. Comput. Secur.*, 20(2–3):259–305, mar 2012.

[BAZB19] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. Cryptology ePrint Archive, Report 2019/191, 2019.

[BBDP01] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In *'01*, Berlin, Heidelberg, 2001.

[BEE+17] Johes Bater, Gregory Elliott, Craig Eggen, Satyender Goel, Abel Kho, and Jennie Rogers. Smcql: Secure querying for federated databases. *Proc. VLDB Endow.*, 10(6):673–684, feb 2017.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. CRYPTO '01, 2001.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS '12*, 2012.

[BHE+18] Johes Bater, Xi He, William Ehrich, Ashwin Machanavajjhala, and Jennie Rogers. Shrinkwrap: Efficient sql query processing in differentially private data federations. *Proc. VLDB Endow.*, 12(3):307–320, nov 2018.

[BK19] Osman Biçer and Alptekin Küpçü. Versatile abs: Usage limited, revocable, threshold traceable, authority hiding, decentralized attribute based signatures. 2019. https://eprint.iacr.org/2019/203.

[BK20] Osman Biçer and Alptekin Küpçü. Anonymous, attribute based, decentralized, secure, and fair e-donation. In *PETS/PoPETS '20*, 2020.

[BKK+21] Dmytro Bogatov, Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. ϵpsolute: Efficiently querying databases while providing differential privacy. In *ACM CCS '21*, 2021.

[BLLN13] Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *Cryptography and Coding*, 2013.

[BM20] Fabio Banfi and Ueli Maurer. Anonymous symmetric-key communication. In *SCN '20*, 2020.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *CRYPTO '12*, 2012.

[BSBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046, 2018. `https://eprint.iacr.org/2018/046`.

[BSCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct Non-Interactive zero knowledge for a von neumann architecture. In *USENIX Sec. '14*, 2014.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *FOCS '11*, 2011.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based fhe as secure as pke. In *ITCS '14*, 2014.

[BYK21] Osman Biçer, Burcu Yıldız, and Alptekin Küpçü. m-stability: Threshold security meets transferable utility. In *ACM CCSW '21*, 2021.

[BZCP14] Yu Bai, Li Zhuo, Bo Cheng, and Yuan Fan Peng. Surf feature extraction in encrypted domain. In *IEEE ICME '14*, 2014.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *CRYPTO '01*, 2001.

[CF13] Dario Catalano and Dario Fiore. Vector commitments and their applications. In *PKC '13*, 2013.

[CGBM15] Henry Corrigan-Gibbs, Dan Boneh, and David Mazieres. Riposte: An anonymous messaging system handling millions of users. In *IEEE S&P '15)*, 2015.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *ASIACRYPT '16*, 2016.

[CJJ+13] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *CRYPTO '13*, 2013.

[CJJ+14] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS '14*, 2014.

[CLT14] Henry Carter, Charles Lever, and Patrick Traynor. Whitewash: Outsourcing garbled circuit generation for mobile devices. In *ACM ACSAC '14*, 2014.

[CXLC14] Fei Chen, Tao Xiang, Xinyu Lei, and Jianyong Chen. Highly efficient linear regression outsourcing to a cloud. *IEEE ToCC*, 2(4):499–508, 2014.

[DCIO98] Giovanni Di Crescenzo, Yuval Ishai, and Rafail Ostrovsky. Non-interactive and non-malleable commitment. In *ACM STOC '98*, 1998.

[DDN91] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. In *ACM STOC '91*, 1991.

[DG03] Ivan Damgård and Jens Groth. Non-interactive and reusable non-malleable commitment schemes. In *ACM STOC '03*, 2003.

[Dia21] Benjamin E. Diamond. Many-out-of-many proofs and applications to anonymous zether. In *IEEE SP '21*, 2021.

[DM15] Léo Ducas and Daniele Micciancio. Fhew: Bootstrapping homomorphic encryption in less than a second. In *EUROCRYPT '15*, 2015.

[DPPS20] Ioannis Demertzis, Dimitrios Papadopoulos, Charalampos Papamanthou, and Saurabh Shintre. SEAL: Attack mitigation for encrypted databases via adjustable leakage. In *USENIX Sec. '20*, 2020.

[Eth] Ethereum. `https://ethereum.org/en/`.

[EZ19] Saba Eskandarian and Matei Zaharia. Oblidb: Oblivious query processing for secure databases. *Proc. VLDB Endow.*, 13(2):169–183, oct 2019.

[Fil] Filecoin. `https://filecoin.io`.

[FLM11] Marc Fischlin, Benoît Libert, and Mark Manulis. Non-interactive and re-usable universally composable string commitments with adaptive security. In *ASIACRYPT '11*, 2011.

[FMMO19] Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. Quisquis: A new design for anonymous cryptocurrencies. In *ASIACRYPT '19*, 2019.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO '86*, 1987.

[Fuj14] Eiichiro Fujisaki. All-but-many encryption: A new framework for fully-equipped uc commitments. In *ASIACRYPT '14*, 2014.

[Fuj16] Eiichiro Fujisaki. Improving practical uc-secure commitments based on the ddh assumption. In *SCN '16*, 2016.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012. `https://eprint.iacr.org/2012/144`.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *ACM STOC '09*, 2009.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO '10*, 2010.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4), 2015.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT '10*, 2010.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO '13*, 2013.

[HL05] Susan Hohenberger and Anna Lysyanskaya. How to securely outsource cryptographic computations. In *TCC '05*, 2005.

[HLH+22] Zhengan Huang, Junzuo Lai, Shuai Han, Lin Lyu, and Jian Weng. Anonymous public key encryption under corruptions. In *ASIACRYPT '22*, 2022.

[HWW+16] Shengshan Hu, Qian Wang, Jingjun Wang, Zhan Qin, and Kui Ren. Securing sift: Privacy-preserving outsourcing computation of feature extractions over encrypted image data. *IEEE TIP*, 25(7), 2016.

[JKPT12] Abhishek Jain, Stephan Krenn, Krzysztof Pietrzak, and Aris Tentes. Commitments and efficient zero-knowledge proofs from learning parity with noise. In *ASIACRYPT '12*, 2012.

[Kup17] Alptekin Küpçü. Incentivized outsourced computation resistant to malicious contractors. *IEEE TDSC*, 14(6), 2017.

[KFTS17] Amrit Kumar, Clément Fischer, Shruti Tople, and Prateek Saxena. A traceability analysis of monero's blockchain. In *ESORICS '17*, 2017.

[Kim20] Jinsu Kim. A post-quantum commitment scheme based on splwe. *IJCSNS*, 20(12):265–271, 2020.

[KS08] Vladimir Kolesnikov and Thomas Schneider. A practical universal circuit construction and secure evaluation of private functions. In *FC '08*, 2008.

[KS16] Ágnes Kiss and Thomas Schneider. Valiant's universal circuit is practical. In *EUROCRYPT '16*, 2016.

[KSN20] Alptekin Küpçü and Reihaneh Safavi-Naini. Smart contracts for incentivized outsourcing of computation. In *ESORICS CBT '20*, 2020.

[KT19] Florian Kerschbaum and Anselme Tueno. An efficiently searchable encrypted data structure for range queries. In *ESORICS '19*, 2019.

[LATV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *ACM STOC '12*, 2012.

[Lav23] Erick Lavoie. Goc-ledger: State-based conflict-free replicated ledger from grow-only counters, 2023.

[LC10] Keng-Pei Lin and Ming-Syan Chen. Privacy-preserving outsourcing support vector machines with random transformation. In *ACM SIGKDD '10*, 2010.

[LJLC12] Jingwei Li, Chunfu Jia, Jin Li, and Xiaofeng Chen. Outsourcing encryption of attribute-based encryption with mapreduce. In *ICICS '12*, 2012.

[LPQ12] Benoît Libert, Kenneth G. Paterson, and Elizabeth A. Quaglia. Anonymous broadcast encryption: Adaptive security and efficient constructions in the standard model. In *PKC '12*, 2012.

[LSP15] Frank Li, Richard Shin, and Vern Paxson. Exploring privacy preservation in outsourced k-nearest neighbors with multiple data owners. In *ACM CCSW '15*, 2015.

[LT22] Zeyu Liu and Eran Tromer. Oblivious message retrieval. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO '22*, 2022.

[Lun18] Joshua Lund. Sealed sender for signal. Technical report, October 2018.

[MGGR13] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *IEEE SP '13*, 2013. `http://zerocash-project.org/media/pdf/zerocash-extended-20140518.pdf`.

[mon] Monero. `https://www.getmonero.org/`.

[MS23] Varun Madathil and Alessandra Scafuro. Prifhete: Achieving full-privacy in account-based cryptocurrencies is possible. Cryptology ePrint Archive, Paper 2023/710, 2023. `https://eprint.iacr.org/2023/710`.

[MSH+18] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, and Nicolas Christin. An empirical analysis of traceability in the monero blockchain. *PETS/PoPETs '18*, 2018.

[MT21] Dimitris Mouris and Nektarios Georgios Tsoutsos. Zilch: A framework for deploying transparent zero-knowledge proofs. *IEEE TIFS*, 16, 2021.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008. `http://bitcoin.org/bitcoin.pdf`.

[NWI+13] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE SP '13*, 2013.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO '91*, 1992.

[PHGR16] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. *CACM*, 59(2):103–112, jan 2016.

[QYL+22] Xuanmei Qin, Zhen Yang, Qi Li, Hongyun Pan, Zhongliang Yang, and Yongfeng Huang. Attribute-based encryption with outsourced computation for access control in iots. In *ACM ASSE '22*, 2022.

[RAD78] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *ACM STOC '05*, 2005.

[Rip] Ripple. `https://ripple.com`.

[SCG+14] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE SP '14*, 2014.

[SS09] Ahmad-Reza Sadeghi and Thomas Schneider. Generalized universal circuits for secure evaluation of private functions with application to data classification. In *Information Security and Cryptology – ICISC 2008: 11th International Conference, Seoul, Korea, December 3-5, 2008, Revised Selected Papers*, pages 336–353, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[ST98] Tomas Sander and Christian F. Tschudin. Towards mobile cryptography. In *IEEE SP '98*, 1998.

[SV10] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *PKC '10*, 2010.

[SYY99] Tomas Sander, Adam Young, and Moti Yung. Non-interactive cryptocomputing for NC^1. *IEEE FOCS '99*, 1999.

[TOR] Tor project. `https://www.torproject.org`.

[TRMP12] Justin Thaler, Mike Roberts, Michael Mitzenmacher, and Hanspeter Pfister. Verifiable computation with massively parallel interactive proofs. In *USENIX HotCloud '12*, 2012.

[DGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT '10*, 2010.

[WBNM22] Chenghong Wang, Johes Bater, Kartik Nayak, and Ashwin Machanavajjhala. Incshrink: Architecting efficient outsourced databases using incremental mpc and differential privacy. In *ACM SIGMOD '22*, 2022.

[WCGFJ12] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *USENIX OSDI '12*, 2012.

[WCH+07] Wai K. Wong, David W. Cheung, Edward Hung, Ben Kao, and Nikos Mamoulis. Security in outsourcing of association rule mining. In *VLDB '07*, 2007.

[WHR+15] Qian Wang, Shengshan Hu, Kui Ren, Meiqi He, Minxin Du, and Zhibo Wang. Cloudbi: Practical privacy-preserving outsourcing of biometric identification in the cloud. In *ESORICS 2015*, 2015.

[WWH+16] Qian Wang, Jingjun Wang, Shengshan Hu, Qin Zou, and Kui Ren. Sechog: Privacy-preserving outsourcing computation of histogram of oriented gradients in the cloud. In *ASIA CCS '16*, 2016.

[XXW13] Xiang Xie, Rui Xue, and Minqian Wang. Zero knowledge proofs from ring-lwe. In *CNS '13*, 2013.

[ZYZL19] Shuoyao Zhao, Yu Yu, Jiang Zhang, and Hanlin Liu. Valiant's universal circuits revisited: An overall improvement and a lower bound. In *ASIACRYPT '19*, 2019.

A Security Analysis of OHE1 and OHE2

In this section, we show the security of OHE1 and OHE2 schemes based on the security of the underlying primitives.

A.1 Proof Sketch of Theorem 1

We fully prove the key privacy and isolation properties of OHE1. We provide proof sketches for full homomorphism and CPA security of OHE1, as the actual proofs are relatively easy, i.e., the former directly follows from the same property of the underlying FHE, and the latter follows from the same property of the underlying FHE and the NIZK functionality.

Lemma 1. *If the underlying FHE scheme has key privacy and CPA security and the underlying (anonymous) commitment scheme is computationally hiding; then the OHE1 scheme has key privacy.*

Proof. The challenger picks $(d_0, d_1) \leftarrow D^2$generates $(pk_0, evk_0, mk_0, sk_0, \pi) \leftarrow \mathsf{OHE1.KeyGen}(params, d_0)$ and
$(pk_1, evk_1, mk_1, sk_1, \pi) \leftarrow \mathsf{OHE1.KeyGen}(params, d_1)$. We would like to prove that

$$\begin{aligned}&\big(pt, pk_0, evk_0, mk_0, pk_1, evk_1, mk_1, \mathsf{OHE1.Enc}_{pk_0}(pt)\big) \approx_c \\ &\quad \big(pt, pk_0, evk_0, mk_0, pk_1, evk_1, mk_1, \mathsf{OHE1.Enc}_{pk_1}(pt)\big).\end{aligned}$$

This is equivalent to proving

$$\begin{aligned}&\big(pt, pk_0, evk_0, mk_0, pk_1, evk_1, mk_1, \mathsf{FHE.Enc}_{pk_0}(pt), \mathsf{FHE.Enc}_{pk_0}(r_0), \mathsf{Commit}(pk_0, r_0)\big) \approx_c \\ &\big(pt, pk_0, evk_0, mk_0, pk_1, evk_1, mk_1 \mathsf{FHE.Enc}_{pk_1}(pt), \mathsf{FHE.Enc}_{pk_1}(r_1), \mathsf{Commit}(pk_1, r_1)\big)\end{aligned}$$

where we dropped the mks as they are composed of pks and and evks. We allow the adversary to choose pt in all hybrids. We start with

Hybrid 0: $\big(pk_0, evk_0, mk_0, pk_1, evk_1, mk_1, \mathsf{FHE.Enc}_{pk_0}(pt), \mathsf{FHE.Enc}_{pk_0}(r_0), \mathsf{Commit}(pk_0, r)\big)$

and then provide intermediary hybrids until obtaining the hybrid
$\big(pk_0, evk_0, pk_1, evk_1, \mathsf{FHE.Enc}_{pk_1}(pt), \mathsf{FHE.Enc}_{pk_1}(r_1), \mathsf{Commit}(pk_1, r_1)\big)$. We show each of them is computationally indistinguishable from the previous one. The transitivity of the computational indistinguishability relation completes this proof.

Hybrid 1: $\big(pk_0, evk_0, mk_0, pk_1, evk_1, mk_1, \mathsf{FHE.Enc}_{pk_1}(pt), \mathsf{FHE.Enc}_{pk_1}(r_0), \mathsf{Commit}(pk_0, r_0)\big)$

Assume there exists a polynomial-time distinguisher $\mathcal{A}$ that can distinguish Hybrid 1 from Hybrid 0 with non-negligible probability. Then, by using $\mathcal{A}$ we can construct a polynomial-time distinguisher $\mathcal{B}$ that can break the key privacy of the FHE scheme. $\mathcal{B}$ deducts (pk_0, evk_0) and (pk_1, evk_1) from the key privacy experiment of the FHE. $\mathcal{B}$ then picks $r_0 \leftarrow R$, and then generates $\mathsf{Commit}(pk_0, r_0)$. It prepares and gives the Hybrid 0 to $\mathcal{A}$. Starting from Hybrid 0, $\mathcal{B}$ continues by generating new sub-hybrids such that each one replaces the next bit ciphertext $\mathsf{FHE.Enc}_{pk_0}(b_i)$ with the challenge in the key privacy experiment of the FHE scheme for $b_i \in (b_1, \ldots, b_{|pt||r_0|}) = pt||r_0$. After each replacement $\mathcal{B}$ asks $\mathcal{A}$ to distinguish the newly generated hybrid from the previous one. Whenever $\mathcal{A}$ can distinguish the new hybrid with non-negligible probability which can only occur due to changing key (otherwise distributions do not change), $\mathcal{B}$ can distinguish the challenged ciphertext with non-negligible probability in the key privacy experiment of the FHE.

Hybrid 2: $\big(pk_0, evk_0, mk_0, pk_1, evk_1, mk_1, \mathsf{FHE.Enc}_{pk_1}(pt), \mathsf{FHE.Enc}_{pk_1}(r), \mathsf{Commit}(pk_0, r_0)\big)$

Assume there exists a polynomial-time distinguisher $\mathcal{A}$ that can distinguish Hybrid 2 from Hybrid 1 with non-negligible probability. Then, by using $\mathcal{A}$ we can construct a polynomial-time distinguisher $\mathcal{B}$ that can break the CPA security of the FHE scheme. $\mathcal{B}$ deducts (pk_1, evk_1) from the key privacy experiment of the FHE (the obtained pk and evk just reassigned). $\mathcal{B}$ first picks $(r_0, r) \leftarrow R^2$, and then generates $\mathsf{Commit}(pk_0, r_0)$. After generating Hybrid 1, $\mathcal{B}$ continues by generating new sub-hybrids such that each one replaces the next bit ciphertext $\mathsf{FHE.Enc}_{pk_1}(b_i)$ with the challenge in the CPA experiment of the FHE scheme. After each replacement $\mathcal{B}$ asks $\mathcal{A}$ to distinguish the newly generated hybrid from the previous one. Whenever $\mathcal{A}$ can distinguish the new hybrid with non-negligible probability which can only occur for ciphertext on a different bit (otherwise distributions do not change), $\mathcal{B}$ can distinguish the challenged ciphertext with non-negligible probability in the CPA experiment of the FHE.

Hybrid 3: $\big(pk_0, evk_0, mk_0, pk_1, evk_1, mk_1, \mathsf{FHE.Enc}_{pk_1}(pt), \mathsf{FHE.Enc}_{pk_1}(r), \mathsf{Commit}(pk_1, r_1)\big)$

Assume there exists a polynomial-time distinguisher $\mathcal{A}$ that can distinguish Hybrid 3 from Hybrid 2 with non-negligible probability. Then, by using $\mathcal{A}$ we can construct a polynomial-time distinguisher $\mathcal{B}$ that can break the computational hiding property of the commitment scheme. Upon obtaining pt from $\mathcal{A}$, $\mathcal{B}$ picks $r \leftarrow R$ and gives (pk_0, pk_1) to the computational hiding experiment challenger. $\mathcal{B}$, in turn, obtains $\mathcal{COMMIT} = \mathsf{Commit}(pk_0, r_0)$ or $\mathcal{COMMIT} \leftarrow \mathsf{Commit}(pk_1, r_1)$. $\mathcal{B}$ itself generates $\mathsf{FHE.Enc}_{pk_1}(pt)$ and $\mathsf{FHE.Enc}_{pk_1}(r)$ and then gives $(pk_0, evk_0,$
$mk_0, pk_1, evk_1, mk_1, \mathsf{FHE.Enc}_{pk_1}(pt), \mathsf{FHE.Enc}_{pk_1}(r), \mathcal{COMMIT})$ to $\mathcal{A}$. Hence, if $\mathcal{A}$ can the new hybrid with non-negligible probability which can only occur for commitment on a different key (otherwise distributions do not change), then $\mathcal{B}$ can use distinguish the challenged $\mathcal{COMMIT}$.

Hybrid 4: $\big(pk_0, evk_0, mk_0, pk_1, evk_1, mk_1, \mathsf{FHE.Enc}_{pk_1}(pt), \mathsf{FHE.Enc}_{pk_1}(r_1), \mathsf{Commit}(pk_1, r_1)\big)$

Assume there exists a polynomial-time distinguisher $\mathcal{A}$ that can distinguish Hybrid 4 from Hybrid 3 with non-negligible probability. Then, by using $\mathcal{A}$ we can construct a polynomial-time distinguisher $\mathcal{B}$ that can break the CPA security of the FHE scheme. $\mathcal{B}$ deducts (pk_1, evk_1) from the key privacy experiment of the FHE (the obtained pk and evk just reassigned). $\mathcal{B}$ first picks $(r_1 \leftarrow R$, and then generates $\mathsf{Commit}(pk_1, r_1)$. After generating Hybrid 3, $\mathcal{B}$ continues by generating new sub-hybrids such that each one replaces the next bit ciphertext $\mathsf{FHE.Enc}_{pk_1}(b_i)$ with the challenger in the CPA experiment of the FHE scheme. After each replacement $\mathcal{B}$ asks $\mathcal{A}$ to distinguish the newly generated hybrid from the previous one. Whenever $\mathcal{A}$ can distinguish with non-negligible probability which can only occur for ciphertext on a different bit (otherwise distributions do not change), $\mathcal{B}$ can use $\mathcal{A}$ to break the CPA security of the FHE scheme.

□

Lemma 2. *If the underlying FHE scheme has weak wrong-key decryption and full homomorphism and the underlying commitment scheme is computationally binding; then the OHE1 scheme has isolation capability.*

Proof. Assume there exists a polynomial-time algorithm $\mathcal{A}$ such that there exists no negligible function $negl(\cdot)$ the inequality

$$Pr\Big[\mathcal{A} \to (ct, e) \text{ s.t. } e \neq \mathsf{OHE1.Dec}_{sk_0}\big(\mathsf{OHE.Mask}_{mk_0}(ct, e)\big), e \neq \mathsf{OHE1.Dec}_{sk_1}\big(\mathsf{OHE.Mask}_{mk_1}(ct, e),\big)\Big] \leq negl(\lambda)$$

holds on keys $(pk_0, evk_0, mk_0, sk_0)$ and $(pk_1, evk_1, mk_1, sk_1)$ generated by the challenger. As $pk_0 \neq pk_1$, the commitment ct_3 of ct can be opened at most one of them, otherwise $\mathcal{A}$ can be used to break binding property of the commitment scheme.

$\mathsf{OHE1.Mask}_{mk_b}$ first recrypts ct_1 and ct_2 to remove the noise and to replace the ciphertexts with $\hat{ct_1}$ and $\hat{ct_2}$ as decryptable by sk_b. If $\hat{ct_1}$ or $\hat{ct_2}$ is not decryptable by sk_b, then $\mathcal{A}$ can be used to break either weak wrong-key decryption or full homomorphism of the FHE scheme as $\mathsf{FHE.Recyrpt}$ algorithm only decrypts homomorphically. That is, if ct_1 or ct_2 is not decryptable then $\mathcal{A}$ can be used for breaking weak wrong-key decryption of the FHE scheme. if ct_1 and ct_2 is decryptable but $\hat{ct_1}$ or $\hat{ct_2}$ is not decryptable, then fully homomorphism property would not hold.

The security of the rest of the $\mathsf{OHE1.Mask}$ operation follows from the full homomorphism of the underlying FHE scheme. We highlight that $\hat{pk}$, $\hat{ct_3}$, and $\hat{parâms_c}$ are computed by the evaluator, so the adversary cannot alter them. Then evaluation of the circuit C_{Commit} homomorphically computes the output of Commit on $\hat{pk_b}$, $\hat{ct_2}$, and $\hat{parâms_c}$. Then the evaluation of $C_{\mathsf{EqualityCheck}}$ compare this with $\hat{ct_3}$. Here, this check can only verify for at most one of the pk_0 and pk_1, as the commitment ct_3 could be made at most one of them. If the result of this check, which is given as the select bit to C_{MUX}, is 0, the output becomes a ciphertext for e. Otherwise, the underlying ciphertext remains unchanged. This is evaluated on the ciphertext $\bar{e}$ generated by the evaluator so the adversary cannot alter it.

□

Lemma 3. *If the underlying FHE scheme has CPA security; then the OHE1 scheme is CPA secure.*

Proof (Sketch). The masking keys mk do not leak any additional information as they are set as (pk, evk). Then, the CPA security of OHE1 is directly based on that of the FHE scheme. Among the ciphertext tuple (ct_1, ct_2, ct_3) generated by the $\mathsf{OHE1.Enc}$ algorithm, only ct_1 contains information on the plaintext pt. So any distinguisher of two ciphertexts in our scheme can be directly used to distinguish those in the FHE scheme, by providing hybrids that alter bit ciphertexts at each step. In fact, the remaining ciphertexts ct_2 and ct_3 can be generated by anyone even the adversary itself. The underlying secrets of remaining ct_2 and ct_3 could even be revealed without breaking CPA.

□

Lemma 4. *If the underlying FHE has full homomorphism; then the OHE1 scheme has full homomorphism.*

Proof (Sketch). Regarding compactness, the freshly generated ciphertexts by $\mathsf{OHE1.Enc}$ only appends the ciphertext ct_2 and ct_3 to the ciphertext ct_1 of the FHE scheme. They have $poly(\lambda)$ and $poly'(\lambda)$ lengths for some polynomials, respectively, independent of the plaintext size. ct_2 has a polynomial-size $poly(\lambda)$ because the randomness used in the commitment scheme must be a polynomial of λ (as the commitment scheme should be computable in polynomial time) and this randomness is then expanded by another polynomial of λ during FHE encryption. ct_3 has a polynomial-size $poly(\lambda)$ because the commitment must be a polynomial of λ (as it should be computable in polynomial time). As the underlying FHE scheme is compact (that is bit encryptions with $\bar{poly}(\lambda)$ size), $|ct|_1$ is $n \cdot \bar{poly}(\lambda)$ for n-bit plaintexts. Hence, we have $n \cdot poly''(\lambda) + poly'(\lambda) + poly(\lambda)$ size ciphertexts, which is $n \cdot \bar{poly}(\lambda)$ for some polynomial $\bar{poly}(\cdot)$. Regarding the ciphertexts obtained by $\mathsf{OHE2.Mask}$ and $\mathsf{OHE2.Eval}$ executions, their compactnesses are also a result of the compactness of the FHE, i.e. ciphertexts have $n \cdot \bar{poly}'(\lambda)$ size. Regarding homomorphism and masked homomorphism, the ciphertexts are indeed obtained as resulting ciphertexts from $\mathsf{FHE.Eval}$ executions. So, they follow from the homomorphism of the FHE scheme. We note that the weak wrong-key decryption property is not needed here, as full homomorphism only deals with correct executions.

□

A.2 Proof Sketch of Theorem 2

We provide sketches of proofs of the key privacy, isolation, CPA security, and full homomorphism of OHE2 as they are similar to those of OHE1. We elaborate on their differences as they slightly change the proofs in Section A.1.

Lemma 5. *If the underlying WWKE scheme has key privacy and CPA security, the underlying FHE scheme has CPA security, and the underlying (anonymous) commitment scheme is computationally hiding; then the OHE2 scheme has key privacy.*

Proof (Sketch). The proof follows the footsteps of the proof of Lemma 1. Instead of the FHE ciphertexts, now we have WWKE ciphertexts. The important difference is the inclusion of tk_0 or tk_1 in the hybrids. Although these are encryptions of the WWKE secret keys under the FHE public keys, thanks to the CPA security of the FHE scheme, we can first replace them with any freshly generated FHE ciphertexts of plaintexts with equal length to WWKE secret keys. Then, the rest of the proof follows the hybrid changes of the proof of Lemma 1, and indistinguishability proofs among them.

□

Lemma 6. *If the underlying WWKE scheme has weak wrong-key decryption, the underlying FHE scheme has full homomorphism, and the underlying commitment scheme is computationally binding; then the OHE1 scheme has isolation capability.*

Proof (Sketch). The proof follows the footsteps as the proof of Lemma 2.Again the binding property of the commitment scheme ensures the commitment ct_3 of a ciphertext can be openable to two different public keys with negligible probability. The main difference from $\mathsf{OHE1.Mask}$ is that instead of $\mathsf{FHE.Recrypt}$ executions, $\mathsf{OHE2.Mask}$ has the homomorphic evaluation of the $C_{\mathsf{WWKE.Dec}}$. Here, similar to the proof of Lemma 2, full homomorphism of the FHE scheme and weak wrong-key decryption property of the WWKE scheme ensures a correct transition. Then, $\mathsf{OHE2.Mask}$ proceeds the same way with homomorphic evaluations as $\mathsf{OHE1.Mask}$ do, so the security follows from the full homomorphism of the FHE scheme.

□

Lemma 7. *If the underlying WWKE scheme has CPA security, and the underlying FHE scheme has CPA security; then the OHE1 scheme is CPA secure.*

Proof (Sketch). The proof follows the footsteps as the proof (sketch) of Lemma 3. Instead of the FHE ciphertexts, now we have WWKE ciphertexts. The important difference is the inclusion of tk in the hybrids. Although this is an encryption of the WWKE secret key under the FHE public key, thanks to CPA security of the FHE scheme, we can first replace it with any freshly generated FHE ciphertext of a plaintext with equal length to a WWKE secret key.

□

Lemma 8. *If the underlying FHE has full homomorphism; then the OHE1 scheme has full homomorphism.*

Proof (Sketch). Regarding compactness of the freshly generated ciphertexts by $\mathsf{OHE2.Enc}$ only appends both the ciphertext ct_2 and ct_3 to the ciphertext ct_1 of the WWKE scheme. They have $poly(\lambda)$ and $poly'(\lambda)$ lengths for some polynomials, independent of the plaintext size. ct_2 has a polynomial-size $poly(\lambda)$ because the randomness used in the commitment scheme must be a polynomial of λ (as the commitment scheme should be computable in polynomial time) and this randomness is then expanded by another polynomial of λ during WWKE encryption (again due to the fact that this algorithm should be computed in polynomial time). ct_3 has a polynomial-size $poly(\lambda)$ because the commitment must be a polynomial of λ (as it should be computable in polynomial time). ct_1 is a WWKE ciphertext with $n \cdot poly''(\lambda)$ size where n is the plaintext size. Hence, we have $n \cdot poly''(\lambda) + poly'(\lambda) + poly(\lambda)$ size ciphertexts, which is $n \cdot \bar{poly}(\lambda)$ for some polynomial $\bar{poly}(\cdot)$. Regarding the ciphertexts obtained by $\mathsf{OHE2.Mask}$ and $\mathsf{OHE12Eval}$ executions, their compactnesses are also a result of the compactness of the

FHE, i.e. ciphertexts have $n \cdot \bar{poly}(\lambda)$ size. Regarding masked homomorphism, the ciphertexts are indeed obtained as resulting ciphertexts from $\mathsf{FHE.Eval}$ executions. So, they follow from the homomorphism of the FHE scheme. We note that the weak wrong-key decryption property is not needed here, as full homomorphism only deals with correct executions.

□