Oblivious Message Retrieval

Zeyu Liu[1] Eran Tromer[1,2]

[1]Columbia University
[2]Tel Aviv University
{zl2967,et2555}@columbia.edu

April 14, 2022

Abstract

Anonymous message delivery systems, such as private messaging services and privacy-preserving payment systems, need a mechanism for recipients to retrieve the messages addressed to them, without leaking metadata or letting their messages be linked. Recipients could download all posted messages and scan for those addressed to them, but communication and computation costs are excessive at scale.

We show how untrusted servers can detect messages on behalf of recipients, and summarize these into a compact encrypted digest that recipients can easily decrypt. These servers operate obliviously and do not learn anything about which messages are addressed to which recipients. Privacy, soundness, and completeness hold even if everyone but the recipient is adversarial and colluding (unlike in prior schemes), and are post-quantum secure.

Our starting point is an asymptotically-efficient approach, using Fully Homomorphic Encryption and homomorphically-encoded Sparse Random Linear Codes. We then address the concrete performance using bespoke tailoring of lattice-based cryptographic components, alongside various algebraic and algorithmic optimizations. This reduces the digest size to a few bits per message scanned. Concretely, the servers' cost is ∼\$1 per million messages scanned, and the resulting digests can be decoded by recipients in under ∼20ms. Our schemes can thus practically attain the strongest form of receiver privacy for current applications such as privacy-preserving cryptocurrencies.

Contents

1 Introduction

End-to-end encryption of message content is well understood and widely practiced. However, metadata about which messages were sent and received by whom, and when, can yield abundant sensitive information via traffic analysis and deductions against auxiliary information. Protecting metadata is thus crucial to anonymous message delivery systems [BLMG21] such as anonymous messaging [WCGFJ12, CGBM15, Lun18], privacy-preserving analytics [BEM$^+$17], and privacy-preserving cryptocurrencies [Noe15, BSCG$^+$14, BCG$^+$20].

Yet, the problem of protecting communication metadata remains an open challenge for many applications, especially when privacy, scalability, efficiency and decentralization are all crucial. This challenge is well exemplified by metadata protection in privacy-preserving cryptocurrencies, such as Zcash [BSCG$^+$14, HBHW21] and Monero [Noe15]. These convey digital asset transaction on a public ledger (blockchain), while keeping the contents of every transaction hidden from all but the counterparties to the transaction (and those they elect to expose it to), using cryptographic protocols utilizing encryption and zero-knowledge proofs. Moreover, the underlying ledger is permissionless, decentralized and widely replicated, allowing anyone to send transactions over the Internet while anonymizing their IP address via standard means such as the Tor network. Supposedly, metadata leaks are thus eliminated.[1]

However, a crucial point lingers. From a receiver's perspective, a transaction pertinent to them could appear anywhere in the ledger. If the receiver has a full copy of the ledger (a "full node" in blockchain parlance), then it could scan it to identify pertinent transactions, but the requisite communication, storage and computation cost may far exceed the capabilities of recipients (e.g., already today, such ledgers are many GB in size; consider, then, wallet apps running on computationally-weak mobile devices, with little storage, using slow or expensive network connections).

How, then, can recipients efficiently *detect* which messages are pertinent to them, and *retrieve* the content of these messages? In general, we consider a bulletin board consisting of numerous messages, with arbitrary application-specific content (of fixed size). Each message is *pertinent* to a single recipient (identified by their public address) to which it was sent, and *impertinent* to other recipients. A recipient, in lieu of receiving and scanning the whole ledger (*full-scan*), may enlist the help of servers, which we call *detectors*, that will help them detect their pertinent messages and retrieve the content of these messages.

One approach is for the recipient to provide the detector with a "detection key" or "incoming viewing key", that allows the detector to check, for each bulletin board message, whether it is pertinent to that recipient [Noe15, HBHW21]. The pertinent messages can then be stored and forwarded to the client. Unfortunately, this exposes metadata to the detector, and thus also to anyone who subverted or coerced the detector, whether in real time or retroactively. Furthermore, such long-lived detection key enables devastating deanonymization attacks.[2]

The problem of *Oblivious Message Detection (OMD)* is to perform such detection without revealing any information to the detectors about which messages are pertinent. This is done

[1]In reality, many of today's blockchain privacy solutions suffer from assorted metadata leaks such as variability in transaction record size, ill-defined cryptographic guarantees, inadequate network-level anonymization, exposure of amounts at the interface between "shielded" and "unshielded" transaction pools, the handling of transaction fees, and common operational mistakes. These are outside our scope.

[2]For example, an adversary who acquired a detection key can easily ascertain whether that key belongs to a given person, by simulating a transaction to that person and seeing if it matches that detection key; if so, then all past and future transaction pertinent to that recipient become linked.

today in Zcash via the ZIP-307 "light client" protocol [GH18], which is essentially optimized full-scan: convert each message to a compact format which contains just enough information for the recipient to check for pertinence, and then send *all* of these compacted messages to the recipient for processing. In practice, this process can take hours even for a relatively lightly-used chain, and is recognized as a severe usability and scalability issue.

Furthermore, the full task is *Oblivious Message Retrieval (OMR)*, where the recipient also gets the content (payload) of their pertinent messages. Given just detection, the recipient would still have to retrieve every detected pertinent message by some means. Naively querying the detector (or some other server) again leaks the pertinency metadata. This could be mitigated by using Private Information Retrieval, mixnets, or decoy traffic, but at high cost or/or ill-defined security. This is recognized by practitioners as an important open problem.[3]

These problems have been studied by two recent works, which made significant headway, but still carry significant drawbacks. Fuzzy Message Detection (FMD) [BLMG21] is based on inducing false-positive decoys into detection, and presents a difficult tradeoff between security (a high decoy density is needed to foil adversarial analysis) and efficiency (these decoys all entail costs). Private Signaling (PS) [MSS$^+$21][4] provides full privacy only if a single detector serves all recipients in the system, and moreover requires either trusted hardware such as Intel SGX, or a pair of servers that are in constant communication but trusted not to collude. Furthermore, both works assume for correctness that all senders and recipients behave honestly, and are susceptible to amplified Denial-of-Service attacks, where an adversary can induce detector and/or recipient work that is disproportional to the number of messages they place on the bulletin board. They also exhibit linkability between detection queries and identities. (See further discussion below.)

We thus pose the problem: *Is it feasible to achieve oblivious message retrieval (and detection) that is fully private, DoS-resistant, unlinkable, trustless, and practical?*

1.1 Our Contributions

In this paper, we propose schemes that fulfill all of the above requirements. Our approach is based on homomorphic encryption using lattice-based cryptography.

Strong Security Definitions. We formally define the notions of Oblivious Message Retrieval and Detection. Our definitions capture natural notions of correctness and privacy, and moreover capture two important security notions that prior works failed to capture or achieve:

- Prior works are vulnerable to *amplified Denial-of-Service* attacks in the realistic threat model where there exist malicious participants (senders or recipients) in the messaging system. Our strengthened notion says that even if arbitrary system participants are adversarial and colluding, they cannot induce more errors or costs than honest participants.

- Prior works are vulnerable to *key-linkability* attacks, which tie retrieval actions to public identities (or to each other) since public keys are themselves reused or linkable. We achieve a strong notion of key unlinkability, preventing these.

[3]E.g., Zcash developers [Hor20] deem it an "action item" that the "lightwalletd [server] learns which transactions belong to the wallet", and accurately described the popular mitigation of using decoy fetches as "security theatre" that fails to achieve unlinkability.

[4]Private Signaling [MSS$^+$21] is a concurrent and independent work.

Possibility of Compact OMR and OMD. We show that Fully Homomorphic Encryption (FHE) can be used to achieve message retrieval and detection with full privacy against any (computationally-bounded) adversary. Moreover, we show that FHE can be used to distill the full bulletin board into a *compact digest* of size that is near-linear in the number of *pertinent* messages, rather than all messages in the bulletin board.

Our approach is based on annotating messages with *clues* to their pertinence, having the detector inspect these clues using FHE and pack the pertinent messages into a compact digest using homomorphically-encoded sparse random linear codes, and having the recipient algebraically reconstruct the messages from the decrypted digest. The homomorphic packing and encoding stages are reminiscent of techniques used in batch Private Stream Search [OS05] and Private Information Retrieval [ACLS18, ALP^{+}21], adapted and optimized to the OMR/OMD setting.

Practical OMR/OMD. Generic use of FHE is notoriously inefficient. We tackle this by a series of optimizations to drastically improve concrete performance. Our techniques include bespoke composition of several different lattice-based schemes (specifically PVW [PVW08] and BFV [Bra12, FV12]) and extensions thereto, utilization of SIMD-like packed operations, using a tailored Sparse Random Linear Code, optimization of multiplicative depth to avoid expensive bootstrapping, and parameter tuning.

We thereby obtain several concrete schemes (summarized in Appendix A),with different trade-offs, that achieve our security notions under standard lattice hardness assumptions (Ring-LWE). Security is thus plausibly postquantum. DoS-resistance holds under an additional natural conjecture about LWE-based encryption, or using zk-SNARKs.

Postquantum Security. Our privacy guarantees are all proven under a standard lattice hardness assumption (Ring-LWE). Therefore, there is no known way by which future quantum computers could retroactively deanonymize transaction recipients using the clues or keys generated in our scheme.[5]

Implementation and Evaluation. We implemented our schemes as an open-source C++ library [OMR21] and measured their concrete performance for a variety of parameters and in comparison to prior work. Salient observations include:

- Detector-to-recipient computation: for Bitcoin-scale parameter settings, our OMR schemes have lower detector-to-recipient communication than any other known retrieval scheme: $\sim$9 bits per bulletin board message for retrieval, and $\sim$4.5 bits/msg for just detection. For even larger parameters, the amortized retrieval digest size drops below 1 bit/msg.[6]
- Recipient's computation is faster than any other known retrieval scheme, e.g., $\sim$20 msec to retrieve 50 pertinent messages out of 500,000.
- Detector's cost for full retrieval is higher than in related schemes, but still quite practical at $\sim$0.065 sec/msg ($\sim$\$1.02 per million messages) on a small cloud VM. For just detection, our scheme is faster than any other known scheme, including those based on trusted hardware.

Thus, our schemes are especially attractive when recipients are limited in bandwidth, computation speed or energy. The one drawback is a one-time cost of uploading a detection key of size $\sim$129 MB to a detector.

[5]Of course, privacy would still be violated if other system components, such as encryption of the message *content* or the *sending* mechanism, are not postquantum-secure.

[6]By comparison, Zcash's ZIP-307 [GH18] uses 928 bits/msg, for just detection.

Scheme	**Privacy**			**Soundness + completeness**	
	Detection	Retrieval	Assumptions	Assumptions	Overflows
Full Scan [GH18]	Full		ECDH + Auth Enc	None	None
FMD1 [BLMG21]	pN-msg-anonymity, fixed $p = 2^{-i}$		PKE	Honest S&R	None
FMD2 [BLMG21]	pN-msg-anonymity, dynamic p		PKE		None
PS1 [MSS+21]	Full	N/A	Trusted Execution Environment (SGX) + DDH + PKE		Undetected
PS2 [MSS+21]	Partitioned across detectors	N/A	Communicating non-colluding servers + Garbled Circuit + Unforgeable Signatures		Undetected
OMRt1 §6.3.2	Full + full-key-unlinkability		FHE		Detected
OMRp1 §7.3	Full + full-key-unlinkability		Ring-LWE	Honest S&R or Conjecture 8.4 or zk-SNARK	Detected
OMRp2 §7.4	Full + full-key-unlinkability		Ring-LWE		Detected

Table 1: Comparison of privacy guarantees and assumptions. Here, pN-msg-anonymity means the recipient's messages are hidden among decoy (false-positive) messages which are a p fraction of the total N messages. "Partitioned across detectors" privacy means that if multiple detectors are used for scalability, then the anonymity set is just the recipients served by the same detector. "Honest S&R" means all senders are honest when generating clue for messages, and all receivers are honest when generating their clue keys. PS1 can be modified for key unlinkability (cf. Section 9.3).

Cryptocurrency Integration. We discuss key design points in integrating our scheme with a blockchain-based privacy-preserving cryptocurrency (exemplified by Zcash) including protocol and costs aspect. We conclude that our scheme is compatible with existing protocols, and that the cost of detection service would be ∼\$1 of Cloud Computing per million messages scanned (i.e., similar magnitude to the total monthly transaction flow in all privacy-preserving cryptocurrencies).

2 Related Work

Privacy-preserving message detection and retrieval has been studied in several prior and concurrent works, discussed below. Tables 1 and 2 summarize functionality, privacy asymptotic aspects, and compare them to our Oblivious Message Retrieval schemes presented in Sections 6 to 9. See Section 10 (e.g., Table 3) for comparison of concrete performance. For reference, these tables also include *full scan*, which is the straightforward linear-communication approach where the recipient scans each message (or a relevant part thereof) in the whole bulletin board (used, e.g., in the Zcash light wallet protocol [GH18]).

2.1 Fuzzy Message Detection

Fuzzy Message Detection (FMD) [BLMG21] addresses the message detection problem. As in our model, senders attach clues to messages in a bulletin board, and detector servers can identify pertinent messages using recipients' detection key. In the following, and in Tables 1 and 2, FMD1 and FMD2 refer to Figures 3 and 4 in [BLMG21], respectively.

Privacy. The privacy notion, which we call *pN-msg-anonymity*, is as follows: the detector can observe which messages were flagged as pertinent, but these are hidden among many intentional false-positives which are indistinguishable from the truly pertinent messages. This is parametrized by a probability $0 < p \leq 1$, such that every message is (mis)detected as pertinent with probability p, resulting in a total of $\sim pN$ decoys for a bulletin board of N messages. As p increases, privacy improves but costs grow, since the recipient needs to receive, and test, all of the decoys.

Scheme	Clue size	Detector cost	Recipient cost	Detection communication	Detection key	Servers	Functionality
Full scan	O(1)	$O(N)$	$O(N)$	$O(N)$	N/A	0	Detect&Retrieve
FMD1	$O(\log(p^{-1}))$	$O(N\log(p^{-1}))$	$O(pN)$	$O(pN)$	$\mathrm{O}(\log(p^{-1}))$	1	Detect&Retrieve
FMD2	$O(\log(p^{-1}))$	$O(N\log(p^{-1}))$	$O(pN)$	$O(pN)$	$\mathrm{O}(\log(p^{-1}))$	1	Detect&Retrieve
PS1	$O(1)$	$O(N\bar{k})$	$O(\bar{k})$	$O(\mathsf{min}(k,\bar{k}))$	O(1)	1	Detect
PS2	$O(1)$	$O(N\bar{k})$	$O(\bar{k})$	$O(\mathsf{min}(k,\bar{k}))$ + s↔s: $O(\bar{k})$	$O(\bar{k})$	2	Detect
OMRt1	$O(\log(\epsilon_{\mathrm{p}}^{-1}))$	$O(N(\log(\hat{k})\log(\epsilon_{\mathrm{n}}^{-1})\log(N)+\log(1/\epsilon_{\mathrm{p}})))$	$O(\hat{k}\log(\hat{k})\log(\epsilon_{\mathrm{n}}^{-1})\log(N)+\hat{k}^3)$	$O(\hat{k}\log(\hat{k})\log(\epsilon_{\mathrm{n}}^{-1})\log(N))$	$O(1)$ (∼16 MB)	1	Detect&Retrieve
OMRp1	$O(\log(\epsilon_{\mathrm{p}}^{-1}))$	$O(N(\log^2(\hat{k})+\log(\epsilon_{\mathrm{p}}^{-1})))$	$O(N+\log^2(\hat{k})\log(\epsilon_{\mathrm{n}}^{-1})+\hat{k}^3)$	$O(N+\log^2(\hat{k})\log(\epsilon_{\mathrm{n}}^{-1}))$	$O(1)$ (∼129 MB)	1	Detect&Retrieve
OMRp2	$O(\log(\epsilon_{\mathrm{p}}^{-1}))$	$O(N(\log(\hat{k})\log(\epsilon_{\mathrm{n}}^{-1})\log^4(N)+\log(1/\epsilon_{\mathrm{p}})))$	$O(\hat{k}\log(\hat{k})\log(\epsilon_{\mathrm{n}}^{-1})\log^4(N)+\hat{k}^3)$	$O(\hat{k}\log(\hat{k})\log(\epsilon_{\mathrm{n}}^{-1})\log^4(N))$	$O(1)$ (∼129 MB)	1	Detect&Retrieve

Table 2: Comparison of asymptotic complexity. Costs are per recipient. N is the total number of messages (in case of PS2, sent to recipients served by that detector.) k is the number of messages pertinent to the specific recipient, and $\bar{k}$ is an upper bound on k provided by the recipient. p is the decoy (false positive) probability of FMD (larger $p \leq 1$ means better privacy). $\hat{k} = \tilde{O}(\bar{k}+\epsilon_{\mathrm{p}}N)$ where ϵ_{n} is the false negative rate and ϵ_{p} is the false positive rate (which can be made polynomially small and does not affect privacy). "s↔s" means the communication cost between two non-colluding detector servers. Payload size and security parameter are taken as constant. Sender's cost is proportional to the clue size in these schemes. OMRt1 is instantiated with TFHE. OMRt1, OMRp1 and OMRp2 are instantiated with SRLC1.

Like other decoy-based privacy notions, pN-msg-anonymity introduces uncertainty into naive analysis, but still allows many privacy-violating deductions, especially given recurring traffic, an active adversary, or small decoy rates p [Lew21a, SPB21].[7]

Retrieval. FMD originally addresses only the detection problem, but it naturally extends to retrieval by attaching the full payload for every detected message; we use this variant when we report the performance of FMD1 and FMD2 as retrieval schemes. Alternatively, retrieval could be done subsequently by some privacy-preserving means, such as PIR (see below).

DoS Attacks and Key Linkability. FMD does not provide soundness guarantees if clues, or clue keys (public addresses), are generated maliciously; it is thus subject to Denial-of-Service (DoS) attacks (see Section 8.2). It also lets detection queries be linked to each other, as well as to clue keys, hence to public identities (see Section 9.1).

2.2 Private Signaling

Another (concurrent and independent) work is Private Signaling (PS) [MSS+21]. Their model assumes that *signals* (analogous to our clues) are sent directly to servers (analogous to our detectors), to which recipients subscribe with their detection keys. Servers handle these signals in a privacy-preserving way, using one of two mechanisms.

The *single-server (PS1)* scheme relies on a Trusted Execution Environment (Intel SGX), running on the detector server, to manage all keys and learn which messages are pertinent to each receiver. This is a very strong trust assumption (especially given the many past attacks on SGX [MSS+21, §2.2.1]), with total privacy failure if violated.

The *two-server (PS2)* scheme instead relies on secure multiparty computation between two servers, using garbled circuit. The two servers jointly serve as a detector, and (shares of) signals are sent to both of them. Privacy holds as long as these servers communicate but do not collude nor leak their secrets. Note that in this model, if either of the servers is compromised and its

[7]For example, the hypothesis that an address belongs to a given user can be tested by observing whether messages sent to that user are *always*, or *rarely*, detected as pertinent under that user's detection key.

data leaks, then the other server can passively deduce the protected information (the recipient of every message), even retroactively. Moreover, unlike some other protocols with non-colluding servers (e.g., information-theoretic PIR), here the two servers are aware of each other's identity and directly communicate, making it easier to perform targeted attacks or induce collusion incentives.

Retrieval. PS addresses only the detection problem, and as for FMD, retrieval needs to be done subsequently by other privacy-preserving means. Unlike FMD, we cannot directly add payloads in plaintext without breaking privacy. Instead, the schemes would need to be nontrivially modified to collect and send the payloads in a privacy-preserving way, with a likely-substantial performance impact. We thus consider PS as a detection-only scheme in Tables 1 to 3, and exclude it from the retrieval comparison of Fig. 3.

Scaling. Both PS schemes are described as having a single detector serving all recipients. However, the workload of the detector is proportional to the number of recipients, limiting scalability. Thus, to compare to our OMD scheme, we consider the scalable generalization where there are multiple detectors, each in charge of some set of users.

For PS1, the generalization to the OMD model is straightforward: the signals can be embedded as clues into the bulletin board messages, and each detector can process the whole bulletin board on behalf of some set of users, preserving full privacy. If any detector's TEE is compromised, then privacy is violated for the recipients served by that detector.

For PS2, such scaling does not work: first, because each detector consists of two servers that must get separate, secret shares of the signal; and second, because the detector's cost is linear in the number of distinct recipient IDs. Therefore, the natural generalization is to partition the network between detectors. Each recipient is served by some detector, and publishes that detector's ID so senders can sent signals to that detector (server pair). Thus, the anonymity set is partitioned: each recipient is anonymous among the users of the same detector. Table 1 reflects these generalizations.

By contrast, our constructions provide full privacy, regardless of how users are split between detectors, and with neither trusted hardware nor non-collusion assumptions.

DoS Attacks and Key Linkability. Like FMD, PS also does not provide soundness guarantees if clues or clue keys are generated maliciously (see Section 8.2).[8] Likewise, it lets detection queries be linked to each other, and to clue keys, hence to public identities (see Section 9.1).

Overflow. PS requires the recipient to set a bound $\bar{k}$ on how many messages it will receive during each period, *in advance* before the period. If more than $\bar{k}$ messages arrive, the PS protocols override old messages, with no indication of overflow. Conversely, our protocol allows $\bar{k}$ to be chosen retroactively (e.g., depending on how much time passed since the previous retrieval), and moreover provides an explicit $\mathsf{overflow}$ indication to the recipient. In PS, the detector's computational cost grows linearly with $\bar{k}$, compared to polylogarithmically in ours.

2.3 Private Retrieval

cPIR. A related problem is *Private Information Retrieval (PIR)* [CGKS95]; and in particular, since we retrieve multiple messages and do not wish to assume non-colluding servers: *multi-query computational PIR (cPIR).*

[8]The threat model in [MSS$^+$21] acknowledges that "we only consider an adversary that aims to break the privacy of the system. For example, corrupt sender could send a malformed location or overload the system with many signals for a particular recipient."

The state-of-the-art multi-query cPIR [ACLS18, ALP$^+$21] assumes that the client/recipient knows the *indices* of the messages to retrieve. It can therefore be used to perform retrieval after the indices have already been detected (i.e., leverage OMD to OMR). However, concrete costs are impractical, in both communication and computation, for applications of interest [Hor21].

Keyword cPIR. Another variant is *multi-query keyword computational PIR*, which allows retrieved messages to be specified by a label, where the label space is potentially much larger from the index space. If these labels can be coordinated between senders and recipients in a privacy-preserving way, then this achieves OMR. Pung [AS16] follows this approach, but requires shared secrets among users, and requires users to participate even if they have nothing to send or receive. It also relies on underlying CPIR, which is again expensive.

Retrieval TEE. Several works [MWS$^+$19, WMS$^+$19, LTHA$^+$20] achieve privacy-preserving retrieval functionality using Trusted Execution Environment, such as Intel SGX. Like cPIR, these can be used to retrieve pertinent messages after their indices have already been detected by OMD. As for PS1 above, TEE is a strong trust assumption, with total privacy failure if violated.

Retrieval via Decoys. In practice, some existing systems implement quasi-private retrieval of messages (after detecting the pertinent ones) by a simple ad hoc attempt to approximate PIR: adding decoy fetches. For example, the Zcash light client prescribes that recipients should "obscure the exact transactions of interest by downloading numerous uninteresting transactions as well" [GH18]. As with the FMD decoys, this provides weak privacy, especially given recurring traffic or an active adversary [Hor20].

Private Stream Search. Private Stream Search (PSS) was introduced by Ostrovsky and Skeith [OS05] and followups [DD07, BSW09, FR13]. It allows a client to search a keyword over a database of documents and download the ones with such a keyword without revealing the keyword to the server.

In terms of techniques, our use of homomorphic accumulation and linear coding is shared also with PSS. However, in PSS, the elements being sought are plaintext words, which allows for relatively simple protocols. In OMR, conversely, the analogues are "clues" which must be randomly sampled and unlinkable, to hide the identity of the recipients. Therefore, we employ FHE to compute a complicated circuit for homomorphic decryption (and amplification) before retrieval, which creates very different cost tradeoffs, optimizations and implementation details compared to PSS. Furthermore, the past PSS works mainly focused on optimizing the communication cost, at the cost of very high server-server computation (e.g., [FR13], using Reed-Solomon coding, has a server perform do computation superlinear in the number of pertinent messages, for every bulletin message); we use different coding techniques to attain practicality in the the OMR setting.

2.4 Other Related Work

The complementary problem of maintaining *sender privacy* when posting on the bulletin board is addressed by Riposte [CGBM15], Signal's Sealed Sender [Lun18] and its improvement [MKA$^+$21].

Alpenhorn [LZ16] addresses privacy-preserving connection establishment. As discussed in [BLMG21], Alpenhorn essentially uses identity-based encryption and a trusted mixnet to reduce that problem to the privacy-preserving message retrieval problem studied by this paper.

The Vuvuzela [JvdHLZZ15] mixnet offers differential privacy for sender and receiver metadata. Rather than a bulletin board model, it assumes an online model where users to remain connected at all times (lest their messages get dropped). It employs a cluster of servers, of which at least one is assumed to be honest. Bandwidth cost is high (e.g, 30 GB/month for users and 416 TB/month for servers [JvdHLZZ15]).

3 Constructions Overview

We provide brief, informal overviews of the constructions described in Sections 6, 7, 8.4 and 9.3.

3.1 OMR1: Compact OMR from generic FHE

As a stepping stone, Section 6 constructs an Oblivious Message Retrieval scheme $\mathsf{OMRt1}$ given any Fully Homomorphic Encryption scheme. It provides full privacy, as well as soundness and completeness in the *non-DoS* threat model defined in Section 4.3. Moreover, this OMR is asymptotically *compact*: the size of the digest, sent from detector to recipient, is quasilinear in the bound $\bar{k}$ on the number of pertinent messages being retrieved, and merely logarithmic in the total number of messages N and the error rate. However, large constants render it impractical by itself.

Setup and sending. Assume we have an FHE scheme, for simplicity over plaintext space $\mathbb{Z}_2$, such as FHEW [DM15] or TFHE [CGGI20]. Each potential recipient generates their own FHE key pair. The FHE public key serves as their clue key, which they publish. Subsequently, to send a message to a recipient, a sender generates a clue consisting of ℓ FHE ciphertexts, all encrypting 1's to the intended recipient's clue key, (where ℓ is logarithmic in the acceptable false-positive rate).

Detection. A recipient contacts a detector and sends their detection key, which is their (rerandomized) FHE public key, along with their chosen bound $\bar{k}$ on the number of messages pertinent to them. The detector also has the full board $\mathsf{BB} = \{(x_1,c_1), \ldots, (x_N, c_N)\}$, consisting of payload-clue pairs.

For each (x_i, c_i), the detector uses the recipient's public key to recrypt (i.e., decrypt under FHE) all the ℓ ciphertexts in the clue c_i. If this message is pertinent to this recipient, then all of the ℓ resulting ciphertexts will encrypt 1, while if the message is impertinent, these ℓ ciphertexts will encrypt random bits (this follows from semantic security). Thus, the detector evaluates the AND of these ciphertexts to generate a pertinency indicator ciphertext PV_i, which encrypts 1 if the message is pertinent, and 0 with probability $1 - 2^{-\ell}$ otherwise. This gives a *Pertinency Vector* PV, of size linear in the total number of messages N.

Compact Detection. To compress PV, the detector uses the following technique. Create $d = \mathsf{poly}(\bar{k})$ buckets. Then, for the i-th message ($i = 1, \ldots, N$), add $i \cdot \mathsf{PV}_i$ (homorphically via a binary circuit) to a randomly-chosen bucket (chosen via a PRG and a random seed s). Then, w.h.p, each bucket would have either no pertinent message index added to it (so it encrypts 0), or one pertinent message index (so it encrypts its index i). In the unlucky case, multiple pertinent messages are added to the same bucket, causing decoding failure; thus duplicate this process several times with fresh (pseudo)random message assignments.

The digest consists of these bucket ciphertexts, along with the seed s. The recipient simply decrypts all buckets to learn the indices of all pertinent messages, w.h.p. unless there were more than $\bar{k}$ messages. To robustly detect overflows, the detector also keeps and sends a global counter that calculates $\sum_{i=0}^{N} \mathsf{PV}_i$ homomorphically via binary circuit. The recipient can decrypt this and verify that all messages were detected.

Compact Retrieval. The detector can moreover compute $\mathsf{ct}_{x_i} = \mathsf{PV}_i \cdot x_i$ homomorphically (via a binary circuit), so that ct_{x_i} encrypts x_i for pertinent messages and 0 for impertinent ones. Then, we use Sparse Random Linear Coding (SRLC) solution as follows. We then create m combinations of these ct_{x_i} by adding them up with different weights ($m = \mathsf{poly}(\bar{k})$). We pseudorandomly assign each of the ct_{x_i} to γ different combinations (for suitable small γ), each with pseudorandom weight w_i, and compute the weighted sum (homomorphically). We then append these combinations to the digest. For the recipient, this yields $m > k$ equations in the k unknown pertinent messages. The right-hand side is formed by the decrypted weighted sums, and the left-hand side is deduced from the indices retrieved from OMD. The recipient can thus use Gaussian elimination to decode the pertinent messages.

This establishes the asymptotic existence of compact OMR (given FHE), but is impractical in computation cost and in clue size.

3.2 OMR2: Practical Non-compact OMR

Sections 7.1 to 7.3 and 7.5 to 7.8 proceed to perform numerous optimizations that improve communication and computation costs to practical levels, as well as security improvements. This results in a new scheme OMRp1, whose digest complexity is not compact (for that, see OMRp2 below), but is concretely about 9 bit/msg for typical parameters. Clue size is reduced to under 1 kB , and the detection time is practical (see Section 10).

OMRp1 achieves full privacy, as well as soundness and completeness, under the standard Ring-LWE hardness assumption. DoS-resistance, under the strong DoS threat model discussed in Section 8, additionally requires a new but natural computational conjecture about Regev05 encryption [Reg09] (i.e., LWE samples), or zk-SNARKs [Mic00, BCCT12].

OMRp1 combines the following optimizations and improvements to OMRt1 (see Fig. 2 for a visual representation of major internal components). It omits the bucket-based packing stage of detection, sending the (packed) PV instead.

Optimization: PVW clue encryption. While the above scheme performs a complex homomorphic operation on the clues, the first operation is always to homomorphically decrypt them within the Recrypt operation. Thus, we don't need to bear the full cost of FHE encryption for the clues. We can use any semantically-secure key-private encryption, and in particular, we choose PVW [PVW08], non-fully-homomorphic LWE-based encryption scheme whose decryption can be evaluated particularly efficiently by FHE schemes such BFV [Bra12, FV12] and BGV [BGV12].

Optimization: Leveled BFV SIMD. We replace generic FHE with BFV [Bra12, FV12] leveled HE (without the expensive Recrypt functionality), which is adequate for decrypting the PVW clues homomorphically and processing PV as above. This requires careful design to minimize the "levels" (i.e., multiplicative depth). Moreover, BFV supports SIMD operations on many plaintext values packed into each ciphertext, which greatly reduces our concrete costs.

Optimization: Non-binary Plaintexts. We generalize the construction from plaintext space $\mathbb{Z}_2$ to $\mathbb{Z}_t$ for prime $t \geq 2$. This has two advantages. First, it lets us attain much better concrete bounds on false positive rate and on SRLC retrieval. And second, it allows us to match the native arithmetic operations of BFV homomorphic operations.

Optimization: PV Packing. We use a dense bit-wise packing of PV. Each BFV ciphertext can pack many bits, and we use each of these to represent an boolean indicator PV_i for $i \in [N]$. Therefore, the digest size is linear in N but very dense: **4.5 bits per message** for the representative parameters of Section 10.

Optimization: Detection Key Size. The use of BFV encryption causes very large detection keys (due to the ciphertexts and evaluation key sizes); we reduce it by packing and by omitting inessential key components.

DoS resistance and Key Unlinkability. The above changes also make it easy to add resistance to Denial of Service attacks (to be discussed in Sections 8 and 9). They also ensure that the detection and clue keys cannot be linked, and can be refreshed whenever desired (Sections 8 and 9).

Streaming Detection Without $\bar{k}$. The majority of the detector's work depends only on the board and detection key, but not on $\bar{k}$. Thus, if $\bar{k}$ is not known in advance (e.g., the recipient does not in advance when is the next time it will connect to the detector to fetch a digest), then the detector can still do most of its work proactively by processing the board messages on the fly. When the recipient eventually shows up and reveals $\bar{k}$, the detector can complete the computation by doing the last step (computing the payload combinations) which is relatively fast, consisting just of homomorphic additions.

Counting Pertinent Transactions. The client can obtain the exact number of pertinent messages (essentially with no extra cost), to robustly detect overflow and know what value of $\bar{k}$ would suffice for repeated retrieval.

3.3 OMR3: Practical Compact OMR

Combining the optimizations of $\mathsf{OMRp1}$ with the compact bucket-based techniques of $\mathsf{OMRt1}$, we obtain $\mathsf{OMRp2}$ in Section 7.4. This is a compact OMR scheme whose digest size is quasilinear in the bound $\bar{k}$ on the number of pertinent messages, and polylogarithmic in the total number of messages N and the error rate. Its constants are mildly larger than $\mathsf{OMRp1}$, making it concretely attractive when $N \gg 10^6$ as seen in Section 10. It also provides full privacy, soundness and completeness in the strong DoS threat model. $\mathsf{OMRp2}$ achieves an amortized digest size of **less than 1 bit per message** for sufficiently large $N \gg \bar{k}$ (see Section 10).

4 Model and Definitions

4.1 System Model

In this section, we define the model and the problem of Oblivious Message Detection and Retrieval. The system components and their high-level properties are as follows (see also Fig. 1).

We have a *bulletin board* (or *board* for short), denoted BB, containing N messages (e.g., blockchain transactions). Each message is sent from some sender and addressed to some recipient, whose identities are supposed to remain private.

A message consists of a pair (x_i, c_i) where x_i is the message *payload* to convey, and c_i is a *clue* string which helps notify the intended recipient (and only them) that the message is addressed to them.

We denote the payload space $\mathcal{P} = \{0{,}1\}^{\tilde{n}}$ for some $\tilde{n} \in \mathbb{Z}^+$, and the clue space $\mathcal{C}$ (typically ℓ number of ciphertexts in some encryption system). The whole board BB (i.e., all payloads and clues) is public. (In applications, the payloads will typically be end-to-end encrypted.)

At any time, any potential recipient p may retrieve the messages addressed to them in BB. We call these messages *pertinent* (to p), and the rest are *impertinent*.

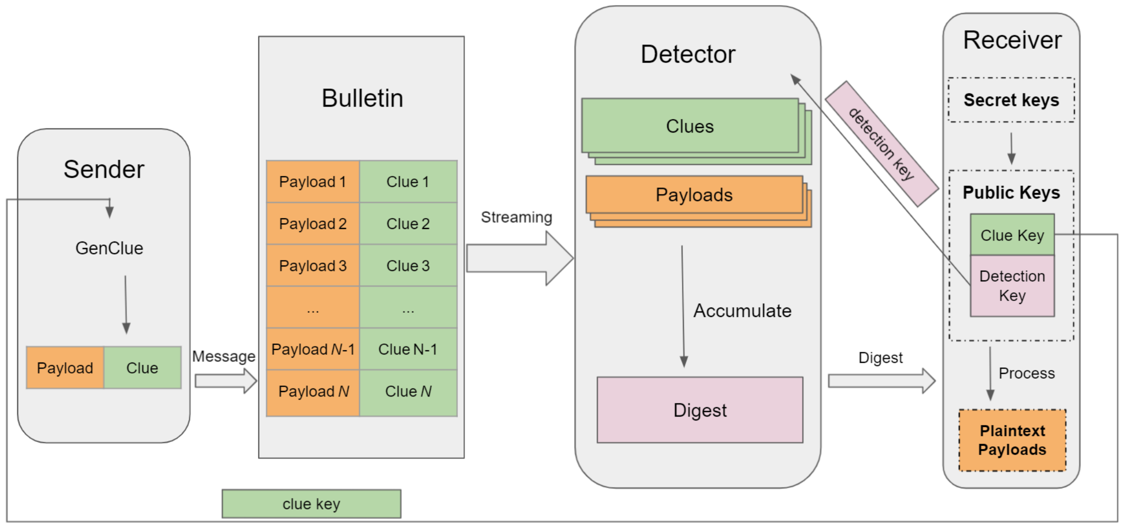


Figure 1: System components

A server, called a *detector*, helps the recipient p *detect* which message indices in BB are pertinent to them, or *retrieve* the payloads of the pertinent messages. This is done obliviously: even a malicious detector learns nothing about which messages are pertinent. The recipient gives the detector their *detection key*, and a bound $\bar{k}$ on the number of pertinent messages they expect to receive. The detector then accumulates all of the messages in BB into string M, called the *digest*, and sends it to the recipient p. The digest M should be much smaller than the board BB (ideally, proportional to $\bar{k}$).

The recipient p processes M to recover all of the pertinent messages with high probability, assuming a semi-honest detector and that the number of pertinent messages did not exceed $\bar{k}$. The *false negative rate* (probability that a pertinent message is not recovered from the digest) is denoted by ϵ_{n}. The *false positive rate* (probability that an impertinent message is output by the recovery procedure) is denoted by ϵ_{p}. Both ϵ_{n} and ϵ_{p} are small (e.g., under 10^{-6}).

There may be many detectors, and each may support many recipients.

Outside our scope are application-specific aspects such as payload encryption, contact discovery and key establishment, the privacy-preserving mechanism by which messages are posted to the board, or how recipients subscribe with detectors. See related discussion in Sections 2 and 11.

4.2 Threat Model (non-DoS)

We assume a computationally-bounded adversary that can read all public information, including all board messages, all public keys in the system, and all communication between the detector and the receiver. It can also honestly generate new messages (with any payload) and post them on the board, as well as honestly generate new clue keys and induce other parties to generate messages addressed to those keys. For soundness and completeness, we require the detectors, senders, and recipients to be honest but curious; they may collude by sharing information. In regard to privacy, we let all parties in the systems be malicious and colluding (including detectors, senders, and recipients), except of course for the sender and recipient of the message(s) whose privacy is to be protected.

A stronger Denial-of-Service (DoS) threat model, which removes the honest-but-curious assumption on message and clue-key generation for soundness and completeness, will be defined in Section 8.1. Additional privacy properties, key unlinkability, will be defined in Section 9.

4.3 Oblivious Message Retrieval

Formally, we capture the notion of an Oblivious Message Retrieval (OMR) scheme as follows.

Definition 4.1 (Oblivious Message Retrieval (OMR))**.** An Oblivious Message Retrieval scheme has the following PPT algorithms:

- $\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n})$: takes a security parameter λ, a false positive rate ϵ_p and a false negative rate ϵ_n, and outputs public parameters pp. pp is implicitly taken by the following algorithms.
- $(\mathsf{sk}, \mathsf{pk} = (\mathsf{pk}_{\mathsf{clue}}, \mathsf{pk}_{\mathsf{detect}})) \leftarrow \mathsf{KeyGen}()$: outputs a secret key sk and a public key pk consisting of a clue key $\mathsf{pk}_{\mathsf{clue}}$ and a detection key $\mathsf{pk}_{\mathsf{detect}}$.
- $c \leftarrow \mathsf{GenClue}(\mathsf{pk}_{\mathsf{clue}}, x)$: takes a clue key and a payload $x \in \mathcal{P}$, and output a clue $c \in \mathcal{C}$.
- $M \leftarrow \mathsf{Retrieve}(\mathsf{BB}, \mathsf{pk}_{\mathsf{detect}}, \bar{k})$: takes as input a board $\mathsf{BB} = \{(x_1,c_1), \ldots, (x_N,c_N)\}$ for some size N, a detection key $\mathsf{pk}_{\mathsf{detect}}$, and an upper bound $\bar{k}$ on the number of pertinent messages addressed to that recipient; and output a digest M.
- $\mathsf{PL} \leftarrow \mathsf{Decode}(M, \mathsf{sk})$: takes the digest M and corresponding secret key sk, and outputs either a decoded payload list $\mathsf{PL} \subset \mathcal{P}^k$ or an overflow indication $\mathsf{PL} = \mathsf{overflow}$.

To define completeness, soundness, and privacy, we first define the notion of board generation:

Definition 4.2 (board generation)**.** Given pp, and N which is the number of messages: Arbitrarily choose the number of recipients $1 \leq p \leq N$, and a partition of the set $[N]$ into p subsets $S_1, \ldots, S_p$ representing the indices of messages addressed to each party. Also arbitrarily choose unique payloads $(x_1, \ldots, x_N)$. For each recipient $i \in [p]$: generate keys $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}()$, and for each $j \in S_i$, generate $c_j \leftarrow \mathsf{GenClue}(\mathsf{pk}_i, x_j)$. Then, output the board $\mathsf{BB} = \{(x_1,c_1), \ldots, (x_N,c_N)\}$, the set S_1, and $(\mathsf{sk}_1, \mathsf{pk}_1 = (\mathsf{pk}_{\mathsf{clue}1}, \mathsf{pk}_{\mathsf{detect}1}))$.[9]

The scheme must satisfy the following properties:

- (Completeness) Let $\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n})$. Set any $N = \mathsf{poly}(\lambda)$, and $0 < \bar{k} \leq N$. Let a board BB, a set S of pertinent messages, and a key pair $(\mathsf{sk}, \mathsf{pk} = (\mathsf{pk}_{\mathsf{clue}}, \mathsf{pk}_{\mathsf{detect}}))$ be generated as in Definition 4.2 for any choice of p, partition and payloads therein. Let $M \leftarrow \mathsf{Retrieve}(\mathsf{BB}, \mathsf{pk}_{\mathsf{detect}}, \bar{k})$ and $\mathsf{PL} \leftarrow \mathsf{Decode}(M, \mathsf{sk})$. Let $k = |S|$ (the number of pertinent messages in S). Then either $k > \bar{k}$ and $\mathsf{PL} = \mathsf{overflow}$, or

$$\Pr[x_j \in Pl \mid j \in S] \geq (1 - \epsilon_\mathrm{n} - \mathsf{negl}(\lambda)) \quad \text{for all } j \in [N] \ .^{10}$$

[9]That is, S_1 is the indices of messages pertinent to the recipient whose keys are $\mathsf{sk}_1, \mathsf{pk}_1$, which wlog is the first recipient.

- (Soundness) For the same quantifiers as in Completeness:

$$\Pr[x_j \in Pl \mid j \notin S] \le (\epsilon_{\mathrm{p}} + \mathsf{negl}(\lambda)) \quad \text{for all } j \in [N] \ .$$

- (Computational privacy) For any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$: let $\mathsf{pp} \leftarrow \mathsf{GenParams}(\epsilon_{\mathrm{p}}, \epsilon_{\mathrm{n}})$, $(\mathsf{sk}, \mathsf{pk} = (\mathsf{pk}_{\mathsf{clue}}, \mathsf{pk}_{\mathsf{detect}})) \leftarrow \mathsf{KeyGen}()$ and $(\mathsf{sk}', \mathsf{pk}' = (\mathsf{pk}'_{\mathsf{clue}}, \mathsf{pk}'_{\mathsf{detect}})) \leftarrow \mathsf{KeyGen}()$. Let the adversary choose a payload x and remember its state: $(x,\mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{pp}, \mathsf{pk}, \mathsf{pk}')$. Let $c \leftarrow \mathsf{GenClue}(\mathsf{pk}_{\mathsf{clue}}, x)$ and $c' \leftarrow \mathsf{GenClue}(\mathsf{pk}'_{\mathsf{clue}}, x)$. Then:

$$\left|\Pr[\mathcal{A}_2(\mathsf{st}, c) = 1] - \Pr[\mathcal{A}_2(\mathsf{st}, c') = 1]\right| \ \le \ \mathsf{negl}(\lambda) \ .$$

An OMR scheme is *compact* if it moreover satisfies the following:

- (Compactness) An OMR scheme is compact if for $\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, \epsilon_{\mathrm{p}}, \epsilon_{\mathrm{n}})$, $(\mathsf{sk}, \mathsf{pk} = (\mathsf{pk}_{\mathsf{clue}}, \mathsf{pk}_{\mathsf{detect}})) \leftarrow \mathsf{OMR.KeyGen}()$, for any board $\mathsf{BB} = \{(x_1,c_1), \ldots, (x_N,c_N)\}$, letting $M \leftarrow \mathsf{Retrieve}(\mathsf{BB}, \mathsf{pk}_{\mathsf{detect}}, \bar{k})$, it always holds that:

$$|M| = \mathsf{poly}(\lambda, \log N) \cdot \log \epsilon_{\mathrm{p}}^{-1} \cdot \tilde{O}(\bar{k} + \epsilon_{\mathrm{p}} N) \ .$$

In the compactness definition, $\tilde{O}(\bar{k} + \epsilon_{\mathrm{p}} N)$ (where $\tilde{O}(x) = x\mathsf{polylog}(x)$) accounts for the number of messages detected as pertinent, including false positives; and the remaining factors account for the cost of representing each such message, taking the payload size as constant.

Adaptively and Maliciously-Generated Boards. The above definition assumes that the board generation is done honestly. This includes the adaptive case, where message recipient and contents are chosen adaptively based on prior messages (which is covered by the universal quantification). However, in real life, clues could be generated maliciously (e.g., not even as a valid $\mathsf{GenClue}$ output). See Section 8 for discussion of real attacks that exploit this, a strengthened definition that achieves *DoS resistance*, and a construction that achieves it.

Repeating Payloads. For simplicity, we considered the case that the payloads $(x_i)_{i \in [N]}$ are all unique (cf. Definition 4.2).[11] Our OMR and OMD definitions naturally extend to the case of duplicate payloads, whether by having PL include the index i of along with every payload x_i, or by expressing pertinence as a multiset of payload (with multiplicity). Our schemes satisfy these generalized definitions.

4.4 Oblivious Message Detection

As a stepping stone towards OMR, we define a weaker functionality, *Oblivious Message Detection (OMD)*, in which the digest merely enables *detection* of the indices of the pertinent messages (for the recipient whose clue key is given), but doesn't convey the corresponding payloads. For brevity, we specify the differences between OMD and OMR.

Instead of a function $\mathsf{Retrieve}(\mathsf{BB}, \mathsf{pk}_{\mathsf{detect}}, \bar{k})$ that enables decoding the pertinent message *payloads*, OMD offers a function $\mathsf{Detect}(\mathsf{BB}, \mathsf{pk}_{\mathsf{detect}}, \bar{k})$ that enables decoding the pertinent message *indices*. The completeness and soundness are changed accordingly:

[10] We denote $[n] = \{1., \ldots, n\}$, see Section 5.1.

[11] In the envisioned applications this is justified since payloads are probabilistically-encrypted ciphertexts and randomized ZK proofs.

- (Completeness) Let $\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n})$. Set any $N = \mathsf{poly}(\lambda)$ and $0 < \bar{k} \leq N$. Let a board BB, a set S of pertinent messages, and a key pair $(\mathsf{sk}, \mathsf{pk} = (\mathsf{pk}_{\mathsf{clue}}, \mathsf{pk}_{\mathsf{detect}}))$ be generated as in Definition 4.2 for any choice of p, partition and payloads therein. Let $M \leftarrow \mathsf{Detect}(\mathsf{BB}, \mathsf{pk}_{\mathsf{detect}}, \bar{k})$ and $\mathsf{PL} \leftarrow \mathsf{Decode}(M, \mathsf{sk})$. Let $k = |S|$ (the number of pertinent messages in S). Then either $k > \bar{k}$ and $\mathsf{PL} = \mathsf{overflow}$, or

$$\Pr[j \in Pl \mid j \in S] \geq (1 - \epsilon_\mathrm{n} - \mathsf{negl}(\lambda)) \quad \text{for all } j \in [N] \ .$$

- (Soundness) For the same quantifiers as in Completeness:

$$\Pr[j \in Pl \mid j \notin S] \leq (\epsilon_\mathrm{p} + \mathsf{negl}(\lambda)) \quad \text{for all } j \in [N] \ .$$

All other interfaces and definitions are the same as OMR.

5 Preliminaries

5.1 Notation

All logarithms are expressed in base 2 if not indicated otherwise. Let $[n]$ denote the set $\{1., \ldots, n\}$, and let $[n,m]$ denote the set $\{n, ..., m\}$. We use $P(\ldots; s)$ to denote a randomized algorithm P running with randomness s. Our big-O notation, when applied to computational and communication complexity, absorbs the security parameter λ and payload size $\tilde{n}$ as a constant.

When we use a pseudorandom function (PRF), we assume that its range is as implied by the context (i.e., that the PRF's outputs are computationally indistinguishable from uniform over that range).

We use $\langle i \rangle_{\mathsf{sk}}$ to denote the space of ciphertexts that decrypt to i under sk. That is, we write $\mathsf{ct} \in \langle i \rangle_{\mathsf{sk}}$ if $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) = i$ with all-but-negligible probability (and where the scheme is not obvious from context, we write: $\langle i \rangle_{\mathsf{sk}}^{\mathsf{scheme}}$).

5.2 LWE Encryption

Our constructions will be optimized by using lattice-based encryption, whose decryption algorithm is particularly simple. We will use the PVW [PVW08] variant of Regev's LWE-based encryption [Reg09], defined as follows.

- $\mathsf{pp}_{\mathsf{LWE}} = (n, \ell, w, q, \sigma) \leftarrow \mathsf{PVW.GenParams}(1^\lambda, \ell, q, \sigma)$: Choose a secret key dimension n, and $w = \mathsf{poly}(\lambda, n, \ell, q)$. Set ciphertext modulus q, number of bits of plaintext modulus, ℓ, and standard deviation for Gaussian distribution for ciphertext noise generation, σ. All parameters n, w, q, σ are chosen as in [PVW08]. $\mathsf{pp}_{\mathsf{LWE}}$ is assumed to be implicitly taken by the following algorithms.

- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{PVW.KeyGen}()$: Choose a secret key $\mathsf{sk} \leftarrow \mathbb{Z}_q^{n \times \ell}$ uniformly at random. Ranomdly sample $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times w}$ and a error matrix $\mathbf{X} \in \mathbb{Z}_q^{\ell \times w}$ from some Gaussian distribution χ_σ, and compute $\mathsf{pk} = (\mathbf{A}, \mathbf{P} = \mathsf{sk}^T\mathbf{A} + \mathbf{X})$.

- $\mathsf{ct} = (\vec{a}, \vec{b}) \leftarrow \mathsf{PVW.Enc}(\mathsf{pk}, \vec{m})$: To encrypt a vector $\vec{m} \in \mathbb{Z}_2^\ell$, define a vector $\mathbf{t} = \frac{q}{2} \cdot \vec{m} \in \mathbb{Z}_q^\ell$. Choose a random vector $\mathbf{e} \leftarrow \{0,1\}^w \in \mathbb{Z}_2^w$ uniformly at random. The ciphertext is the pair $(\vec{a}, \vec{b}) = (\mathbf{Ae}, \mathbf{Pe} + \mathbf{t}) \in \mathbb{Z}_q^n \times \mathbb{Z}_q^\ell$.

- $\vec{m} \leftarrow \mathsf{PVW.Dec}(\mathsf{sk}, ct = (\vec{a},\vec{b}))$: $\vec{d} = \vec{b} - \mathsf{sk}^T\vec{a} \in \mathbb{Z}_q^\ell$, $\vec{m} = \lfloor \frac{\vec{b}+q/4}{q/2} \rfloor \in \mathbb{Z}_2^\ell$

PVW is unconditionally correct (sound), and under the standard *Learning With Error (LWE)* hardness assumption [Reg09, APS15] it fulfills the standard definitions of semantic security (IND-CPA) and key privacy.

5.3 Homomorphic Encryption

Fully Homomorphic Encryption (FHE), introduced by Rivest et. al. [RAD78] and first accomplished by Gentry in [Gen09], enables evaluation of a circuit on encrypted data, such that the resulting ciphertext (when decrypted using the private key) is the output of the circuit on the data, but the evaluator learns nothing about the data or the output.

Formally, an (asymmetric) FHE scheme is an encryption with PPT algorithms $\mathsf{GenParams}(1^\lambda)$, $\mathsf{KeyGen}()$, $\mathsf{Enc}(\mathsf{pk}, m)$, $\mathsf{Dec}(\mathsf{sk}, c)$ fulfilling the standard definitions of semantic security, soundness, and key privacy. Moreover, it has two more PPT algorithms: $\mathsf{Eval}(\mathsf{pk}, (\mathsf{ct}_1, \ldots, \mathsf{ct}_k), C)$, $\mathsf{Recrypt}(\mathsf{pk}, \mathsf{ct})$. $\mathsf{Recrypt}$ function is as defined in [Gen09], i.e., it decrypts homomorphically using the encrypted secret key, thus yielding a fresh re-encryption of the original plaintext. These two new algorithms fulfill the following generalized correctness. Given a circuit C and plaintexts $(m_1, ..., m_k)$, ciphertexts $(ct_1, ..., ci_k)$ which are each either $\mathsf{ct}_i \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, m_i)$ or $\mathsf{ct}_i \leftarrow \mathsf{FHE.Recrypt}(\mathsf{pk}, \mathsf{FHE.Enc}(\mathsf{pk}, m_i))$, and letting $\mathsf{ct}' \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, C, (\mathsf{ct}_1, ..., \mathsf{ct}_k))$: $\Pr[\mathsf{FHE.Dec}(\mathsf{sk}, ct') = C(m_1, ..., m_k)] \geq 1 - \mathsf{negl}(\lambda)$.

We require an additional property for the FHE scheme:

Definition 5.1 (Wrong-Key Decryption)**.** For an FHE scheme with plaintext space $\mathbb{Z}_t$, and $t \geq 2$, letting $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$ and $(\mathsf{sk}', \mathsf{pk}') \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$, $\mathsf{ct} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, 1)$, and $m' \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}', \mathsf{FHE.Recrypt}(\mathsf{pk}', ct))$, it holds that: $\Pr[m' = 1] \leq 1/p + \mathsf{negl}(\lambda)$.

5.3.1 Brakerski/Fan-Vercauteran Homomorphic Encryption

We use the Brakerski/Fan-Vercauteran homomorphic encryption scheme [Bra12, FV12], which we refer to as the BFV scheme. Given a polynomial from the cyclotomic ring $R_t = \mathbb{Z}_t[X]/(X^D + 1)$, the BFV scheme encrypts it into a ciphertext consisting of two polynomials, where each polynomial is from a larger cyclotomic ring $R_q = \mathbb{Z}_q[X]/(X^D + 1)$ where $q > t$. We refer to t, q, and D as the plaintext modulus, the ciphertext modulus, and the ring dimension, respectively.

Each ciphertext can pack D plaintext group elements $(m_1, \ldots, m_D) \in \mathbb{Z}_t^D$, and "single instruction, multiple data" (SIMD) homomorphic operations can be applied to these.

BFV is unconditionally correct (sound), and under the standard *Ring-LWE (RLWE)* [LPR13, Pla18] hardness assumption it fulfills the standard definitions of semantic security (IND-CPA).

Our practical schemes will use BFV as a leveled homomorphic encryption (i.e., applied to circuits of bounded multiplicative depth), without invoking $\mathsf{Recrypt}$.

6 Generic OMR and OMD Using FHE

In this section, we discuss how to construct Oblivious Message Retrieval (OMR). We start by constructing Oblivious Message Detection (OMD), and then extend it to OMR. This section's constructions will be based on general fully homomorphic encryption (FHE), and assume all the clues are generated honestly.

Clueless FHE-Based OMD. Before proceeding, we observe that in principle there is a conceptually straightforward approach to OMD. Presumably, the anonymous message delivery system already has some predicate which a recipient can apply to a candidate message, together with their secret data, to check if the message is pertinent to them (e.g., by trial decryption). Thus, we can have the detector homomorphically evaluate that predicate under FHE on every message, given a detection key which is the recipient's secret data encrypted under FHE. Then, homomorphically pack the result into a digest which contains the indices of pertinent messages, still encrypted. The recipient merely decrypts this FHE ciphertext. This does not even require any new clues or clue keys, since it reuses whatever means the system already has to define pertinence. Alas, for typical protocols this would be completely impractical.[12] Thus, our schemes will add dedicated clues that are specifically crafted to be easy to process under homomorphic encryption.

Stepping-stone FHE-Based OMD. The following generic FHE-based approach serves as a stepping stone to the improved ones, and a simple proof of theoretical possibility. It is still not directly practical. In Section 7 and later, we improve it to achieve practicality and strengthen its security guarantees.

6.1 Oblivious Message Detection Using FHE

6.1.1 Non-compact Construction of OMD

We start by showing how pertinent messages can be *detected*, with a small digest, using any key-private public-key FHE satisfying Definition 5.1 (e.g, FHEW/TFHE [DM15, CGGI20][13]). For simplicity, assume the plaintext space is $\mathbb{Z}_2$.[14] Our didactic starting point is a straightforward, non-compact construction; we will then improve it to achieve compact detection and retrieval.

High-Level Idea. Each clue c_i for $i \in [N]$ consists of ℓ ciphertexts, each encrypting the constant 1 under the public key of the party this message is addressed to. The detector, serving recipient p, will use p's FHE public key pk to recrypt all the ciphertexts in all the clues: $c_{i,j} \mapsto c'_{i,j}$ for $j \in [\ell]$. Crucially, note that each such $c'_{i,j}$ will be $\langle 1 \rangle_{\mathsf{sk}}$ if the message is addressed to p, and otherwise be 1 with probability $\leq 1/2 + \mathsf{negl}(\lambda)$ when decrypted by sk, the corresponding secret key for the recipient p (by Definition 5.1). Thus, for each message, The detector performs an AND gate over all $c'_{i,j}$ ($j \in [\ell]$) to get PV_i (the *Pertinency Vector*) for all $i \in [N]$. Then the detector sends PV vector to the recipient. The recipient decrypts each PV_i using its FHE secret key, and if the result is 1, deems the i-ith message pertinent.

Theorem 6.1. *The scheme* $\mathsf{OMDt1}$ *in Algorithm 1 is an Oblivious Message Detection scheme, when instantiated with any fully homomorphic encryption scheme* FHE.

Proof. Completeness: If a clue $c = (c_1, \dots, c_\ell)$ was encrypted to the clue key pk of a recipient p, all of the $\mathsf{FHE.Recrypt}(c_i, \mathsf{pk})$ will output $\langle 1 \rangle_{\mathsf{sk}}$ with overwhelming probability, and thus the AND

[12]E.g., in Zcash, the predicate includes elliptic curve Diffie-Hellman key agreement [HBHW21, §4.17.2], which would be very expensive under known FHE schemes.

[13]Key privacy for these two schemes is the same as the LWE-based Regev05 encryption on which they are based. Similarly property in Definition 5.1 holds because their $\mathsf{Recrypt}$ function is simply homomorphically perform the Regev05 decryption.

[14]The following naturally generalizes to plaintext space $\mathbb{Z}_t$ for any prime t. For brevity, we kept this section focused on $\mathbb{Z}_2$, which suffices for its results, and added footnotes to clarify the generalization. Section 7 will use $t > 2$.

Algorithm 1 Simple Non-compact Oblivious Message Detection

```
1: procedure OMDt1.GenParams($1^\lambda, \epsilon_p, \epsilon_n$)
2:     Let $\ell$ be the smallest integer s.t $2^{-\ell} \leq \epsilon_p$
3:     $\mathsf{pp_{FHE}} \leftarrow \mathsf{FHE.GenParams}()$
4:     return $\mathsf{pp} = (1^\lambda, \ell, \epsilon_p, \epsilon_n, \mathsf{pp_{FHE}})$
5: procedure OMDt1.KeyGen
6:     $(\mathsf{FHE.sk}, \mathsf{FHE.pk}) \leftarrow \mathsf{FHE.KeyGen}(\mathsf{pp_{FHE}}, \lambda)$
7:     return $(\mathsf{sk} = \mathsf{FHE.sk}, (\mathsf{pk_{clue}} = \mathsf{FHE.pk}, \mathsf{pk_{detect}} = \mathsf{FHE.pk}))$
8:                          ▷ for key unlinkability (cf. §9.3): $\mathsf{pk_{clue}} \leftarrow \mathsf{FHE.RegenPK}(\mathsf{FHE.sk})$
9: procedure OMDt1.GenClue($\mathsf{pk_{clue}}, \epsilon_p, \epsilon_n, x$)
10:    $c_i \leftarrow \mathsf{FHE.Enc}(\mathsf{pk_{clue}}, 1)$ for all $j \in [\ell]$
11:    return $(x, c = (c_j)_{j\in[\ell]})$
12: procedure OMDt1.Detect($\mathsf{BB} = ((x_i, c_i)_{i\in[N]}), \mathsf{pk_{detect}}, \bar{k}, \epsilon_n$)
13:    for $i = 1$ to $N$ do
14:        for $c_{i,j} \in c_i = (c_{i,j})_{j\in[\ell]}$ do
15:            $c'_{i,j} \leftarrow \mathsf{FHE.Recrypt}(\mathsf{pk_{detect}}, c_{i,j})$
16:        $\mathsf{PV}_i \leftarrow \mathsf{FHE.Eval}((c'_{i,j})_{j\in[\ell]}, \mathsf{AND})$
17:    return $M = (\mathsf{PV}_i)_{i\in[N]}$
18: procedure OMDt1.Decode($M = (\mathsf{PV}_i)_{i\in[N]}, \mathsf{sk}$)
19:    for $c_{i,j} \in c_i = (c_{i,j})_{j\in[\ell]}$ do
20:        If $\mathsf{FHE.Dec}(\mathsf{PV}_j, \mathsf{sk}) = 1$: append $j$ to $\mathsf{PL}$
21:    return $\mathsf{PL}$
```

gate will output $\langle 1\rangle_{\mathsf{sk}}$, which the detector will decode as 1, indicating a pertinent message. False negatives are induced just by the (negligible) correctness error of the underlying FHE scheme.

Soundness: If a clue $c = \{c_i\}_{i\in[\ell]}$ is encrypted to the clue key of a party other than p, then $\mathsf{FHE.Recrypt}(c_i, \mathsf{pk})$ will output $\langle 1\rangle_{\mathsf{sk}}$ with probability only (roughly) half by Definition 5.1. If any of the result is $\langle 0\rangle_{\mathsf{sk}}$, the AND gate would result in $\langle 0\rangle_{\mathsf{sk}}$. Therefore, the decryption would be 0 with probability $1 - 2^{-\ell}$ and the soundness follows.

Privacy: Privacy is implied by the key-privacy of the underlying FHE scheme (cf. Section 5.3). Clues contains just ciphertexts encrypted to the recipient's clue key (i.e., FHE public key), which are indistinguishable from those encrypted to other FHE public keys. □

Complexity. The digest size is linear in the number of messages in board, $|M| = O(N)$. The computational complexity for the detector is $O(N \log(\epsilon_p^{-1}))$, since each message has $O(\log(\epsilon_p^{-1}))$ ciphertexts. The recipient computational complexity is $O(N)$, for decrypting the digest.[15]

6.1.2 Compact Construction of OMD

Our next step is to achieve compactness, i.e., a digest size which depends primarily on the number of *pertinent* messages k, rather than the total number of messages N. Since k itself is a private

[15]Here and elsewhere in this paper, we consider the security level λ and payload size $\tilde{n}$ to be absorbed by the big-O notation. This lets us focus on how costs scale for a given application and security level.

information that should not be exposed, the digest size (and detector's computation) instead need to depend on an *upper bound* $\bar{k}$ on the number of pertinent messages k for the particular recipient (see Definition 4.1). The bound $\bar{k}$ is given as a parameter to the detector.[16]

The technique used here is reminiscent of that used in [OS05], where homomorphic operations are used to summarize "documents" according to 0/1 encryptions; in our case the the 0/1 encryptions are indirectly derived by homomorphic computation (as above), and the (implicit) "documents" are our message indices.

High-Level Idea. We start as above, with the same clue keys, and with the detector recrypting these and computing PV_i ciphertexts which are $\langle 1 \rangle_{\mathsf{sk}}$ iff the i-th message is pertinent w.h.p. The detector then compresses the PV information as follows.

We create m *buckets*, for some $m > \bar{k}$ (where $m = O(\mathsf{poly}(\bar{k}))$, to be fixed later). Each bucket contains an *accumulator* Acc_i containing a value in $\mathbb{Z}_N$ encrypted under pk, represented as a vector of $\lceil \log(N) \rceil$ ciphertexts of FHE that store the bit-wise binary encoding of the $\mathbb{Z}_N$ element.[17] A bucket also contains a *counter* $\mathsf{Ctr}_{i \in [m]}$, where Ctr_i contains a value in $\mathbb{Z}_{\bar{k}}$ encrypted under pk, represented as a vector of $\lceil \log(\bar{k}) \rceil$ ciphertexts that contains the bit-wise binary encoding. All buckets and counters are initialized to encryptions of zero.

To add the i-th message in the board, the detector computes PV_i as in Algorithm 1, draws a random bucket index $\mu \in [m]$, and homomorphically adds PV_i to counter Ctr_μ in $\mathbb{Z}_{\bar{k}}$. It also homomorphically computes PV_i and adds it to the accumulator Acc_μ.

Finally, the detector sends all buckets and counters to the recipient. The recipient decrypts these and checks: if the decrypted Ctr_μ is 1, then bucket has a single pertinent message mapped to it, and the decrypted Acc_μ gives the index of that message. If any Ctr_μ decrypts to greater than 1, indicating that several pertinent messages collided in the μ-th bucket, then the detection fails and outputs $\mathsf{overflow}$.

This method gives us a success rate of $\rho = \prod_{i=1}^{\bar{k}-1} \frac{(m-i)}{m}$. To achieve the desired ϵ_{n}, we amplify by repeating this C times with fresh buckets and counters and re-randomized assignment of messages to buckets, such that $\rho^C < \epsilon_{\mathrm{n}}$. [18]

Gathering Partial Information. The amplification can be optimized as follows. Even when a set of buckets contains collisions and thus doesn't directly decode to give all pertinent message indices, there will likely be *some* buckets in the set that do yield useful information (i.e., the corresponding counters are $\langle 1 \rangle_{\mathsf{sk}}$). If we gather all such partial information together, we may get the full information. Then, the failure probability is $< 1 - \prod_{i=1}^{\bar{k}-1}(1 - (\frac{i}{m})^C)$.[19]

Parameter Analysis. The scheme requires choice of m and C. If we choose $m = 10\bar{k}$, we can then choose the smallest integer C such that $1 - \prod_{i=1}^{\bar{k}-1}(1 - (\frac{i}{10\bar{k}})^C) \leq \epsilon_{\mathrm{n}}$. We can see that $C = O(\log(\bar{k}) \log(1/\epsilon_{\mathrm{n}}))$ by using union bound. We get similar results for any choice of $m > k$ that is linear in k.

Decoding k. We can let the recipient learn the actual number k of pertinent messages (even if full decoding fails due to unresolved collisions), by adding another global counter for the total number

[16] If the actual number of pertinent messages k exceeds the assumed bound $\bar{k}$, then retrieval may fail. The recipient can detect such overflow and ask for the detection to be redone with a larger $\bar{k}$. Our scheme gives the client the exact number of k, as discussed below.

[17] For FHE over $\mathbb{Z}_t$, use $\lceil \log_t(N) \rceil$ ciphertexts per bucket.

[18] For example, with $\bar{k} = 10$, $m = 100$, and $C = 15$, we get a failure probability $< 2^{-20}$.

[19] To achieve failure probability of $< 2^{-20}$ for $\bar{k} = 10$, we then only need $m = 50, C = 9$, a threefold reduction in digest size.

of pertinent messages, Ttl, represented as a vector of $\lceil\log(N)\rceil$ ciphertexts of FHE containing the bit-wise binary representation of $k \in \mathbb{Z}_N$. The detector homomorphically sums all PV's into Ttl.

Theorem 6.2. *The scheme* $\mathsf{OMDt2}$ *in Algorithm 2 is a compact Oblivious Message Detection scheme, when instantiated with any fully homomorphic encryption scheme* FHE *and a PRF* f*.*

Proof sketch . Soundness: Follows from that of $\mathsf{OMDt1}$. The detector would output an index for which $\mathsf{PV}_i \notin \langle 1\rangle_{\mathsf{sk}}$ only in case of incorrect FHE decryption, which has negligibly probability.

Completeness: Assume, initially, that there is no false positives. Then completeness of $\mathsf{OMDt1}$ ensures that pertinent messages i have $\mathsf{PV}_i = \langle 1\rangle_{\mathsf{sk}}$. Random assignment gives ϵ_{n} failure probability given suitable parameter choice (line 6) when $k \leq \bar{k}$. Our assignments are *pseudo*random, which has negligible influence. Therefore, for $k \leq \bar{k}$, we have failure probability $< \epsilon_{\mathrm{n}} + \mathsf{negl}(\lambda)$. If $k > \bar{k}$, the error probability may be higher (and eventually 1 because there is insufficient room in the digest for all pertinent messages), but the recipient learns k and is allowed to just output $\mathsf{overflow}$ in this case.

Excessive false positives could break completeness by overflowing the buckets. However, the probability of having more than $N\log(N)\epsilon_{\mathrm{n}}$ false positives is negligible, by soundness. Thus we increased $\bar{k}$ to $\hat{k} \leftarrow \bar{k} + N\log(N)\epsilon_{\mathrm{n}}$ (line 3), making the above argument apply.

Privacy: Follows from that of $\mathsf{OMDt1}$, since no additional values are sent by the message sender or recipient.[20]

Compactness: Communication cost is then $O(\hat{k}\log(\hat{k})\log(\epsilon_{\mathrm{n}}^{-1})\log(N))$, where $\hat{k} = \tilde{O}(\bar{k} + N\epsilon_{\mathrm{p}})$. $mC = O(\hat{k}\log(\hat{k})\log(\epsilon_{\mathrm{n}}^{-1}))$ is the number of buckets as we analyzed above, and $\log(N)$ is the number of ciphertexts in each bucket. $\square$

Complexity. The detector's computational complexity is quasilinear in N and sublinear in $\bar{k}$: $O(N(\log(\hat{k})\log(\epsilon_{\mathrm{n}}^{-1})\log(N) + \log(1/\epsilon_{\mathrm{p}})))$. Indeed, each message requires recryption of the $\log(1/\epsilon_{\mathrm{p}})$ ciphertexts in the clue, and adding pertinent indices to buckets (C copies for each message), each containing $\log(N)$ ciphertexts.

The recipient's computational complexity is $O(\hat{k}\log(\hat{k})\log(\epsilon_{\mathrm{n}}^{-1}) + \hat{k}^3)$ where the first term for decryption (thus proportional to the size of digest), and the second term is for Gaussian elimination.

This construction satisfies the asymptotic requirements of compact OMR (completeness, soundness, privacy, and compactness). The above is still far from concrete practicality (see end of Section 6), as will be addressed in Section 7.

Optimizations and Alternative Constructions. Appendices B.1 and B.2 describes an optimization that resolves collisions after gathering partial information. It also describes two alternative constructions. The first homomorphically searches for an empty bucket in which to put each new pertinent. The second homomorphically sorts the values $\mathsf{PV}_i \cdot i$, leading the pertinent messages to the top of the sorted list, which can then be used as the digest.

6.2 Payload Retrieval using FHE

An OMD scheme, like the one above, lets the recipient learn the *indices* of the messages addressed to it in the board. It would then still need to retrieve the *content* (or *payload*) of those messages,

[20] Note that in our model, the detector does not learn whether retrieval succeeded. Knowing the latter would reject some hypotheses about which message combination are pertinent. We discuss this further in Section 7.7

Algorithm 2 Simple Compact Oblivious Message Detection

```
1: Procedures OMDt2.GenParams, OMDt2.KeyGen and OMDt2.GenClue are identical to OMDt1.
2: procedure OMDt2.Detect(BB = $((x_i, c_1)_{i\in[N]})$, pk, $\bar{k}$)    ▷ Implicitly taking pp = $(\ell, \epsilon_p, \epsilon_n, \mathsf{pp}_{\mathsf{FHE}})$
3:     $\hat{k} \leftarrow \bar{k} + N\log(N)\epsilon_p$
4:     $m \leftarrow 10\hat{k}$    ▷ Asymptotically, $m$ can be $c \cdot k$ for any $c > 1$ (see Parameter Analysis)
5:     Draw a random PRF seed $s$
6:     Let $C$ be the smallest integer such that $1 - \prod_{i=1}^{\hat{k}-1}(1-(\frac{i}{m})^C) < \epsilon_n$
7:     for $o = 1$ to $C$ do
8:         Initialize Acc = $(\mathsf{Acc}_j = (\mathsf{FHE.Enc}(\mathsf{pk}_{\mathsf{detect}}, 0^{\lceil\log(N)\rceil})))_{j\in[m]}$
9:         Initialize Ctr = $(\mathsf{Ctr}_j = (\mathsf{FHE.Enc}(\mathsf{pk}_{\mathsf{detect}}, 0^{\lceil\log(\hat{k})\rceil})))_{j\in[m]}$
10:        Ttl $\leftarrow$ FHE.Eval(pk, $(\mathsf{PV}_i)_{i\in\lceil\log(N)\rceil}$, +)
11:        for $i = 1$ to $N$ do
12:            $\ell \leftarrow |c_i|$
13:            for $c_{i,j} \in c_i = (c_{i,j})_{j\in[\ell]}$ do
14:                $c'_j \leftarrow$ FHE.Recrypt(pk, $c_{i,j}$)
15:            $\mathsf{PV}_i \leftarrow$ FHE.Eval($\{c'_j\}_{j\in[\ell]}$, AND)
16:            $v \leftarrow f_s(j, o) (\mathrm{mod}\ m)$    ▷ pick a bucket pseudorandomly
17:            tmp $\leftarrow$ FHE.Eval($\mathsf{PV}_i$, $j$, $\times$)
18:            $\mathsf{Acc}_v \leftarrow$ FHE.Eval($\mathsf{Acc}_v$, tmp, +) over $\mathbb{Z}_N$    ▷ through binary circuit
19:            $\mathsf{Ctr}_v \leftarrow$ FHE.Eval($\mathsf{Ctr}_v$, tmp, +) over $\mathbb{Z}_{\hat{k}}$
20:        $M_o$ = (Acc, Ctr)
21:    return $M = (M_o)_{o\in[C]}$
22: procedure OMDt2.Decode($M = (M_o)_{o\in[C]}$, sk)    ▷ Implicitly taking pp = $(\ell, \epsilon_p, \epsilon_n, \mathsf{pp}_{\mathsf{FHE}})$
23:    $k \leftarrow$ FHE.Dec(sk, Ttl)
24:    for $o = 1$ to $C$ do
25:        (Acc, Ctr) $\leftarrow M_o$
26:        for $i = 1$ to $N$ do
27:            If FHE.Dec(sk, $\mathsf{Ctr}_i$) = 1: add FHE.Dec(sk, $\mathsf{Acc}_i$) to the set PL
28:    If $|Pl| \neq k$, PL = overflow
29:    return PL
```

and (in our motivating applications) do so privately without revealing which messages were of interest. As discussed in Section 2, retrieval with current methods *after* obtaining the indices is either inefficient or non-private.

We thus extend the above OMD scheme to Oblivious Message Retrieval, starting with a simple construction from generic FHE below, and improving it in Section 7. Our approach embeds techniques from PSS [OS05, DD07, BSW09, FR13] and multi-query PIR [ACLS18, ALP$^+$21] (see Section 2.3) into the detector's operation, without an extra round-trip to the client. Specifically, we use the encrypted 0/1 pertinency bits, derived above, to extract the pertinent payloads using homomorphic multiplication; we then compress these using homomorphically-evaluated linear codes.As discussed in Section 2.3, these techniques require substantial adaptations for efficiency in OMD/OMR.

The generic OMR approach is detailed below, first as an inefficient construction based on generic FHE (as a didactic stepping stone), and then with major optimizations (Section 7).

Single Pertinent Message. Our starting point is the above $\mathsf{OMDt1}$ construction in Algorithm 1. The detector's procedure $\mathsf{OMR.Retrieve}$ first runs $\mathsf{PV} \leftarrow \mathsf{OMDt1.Detect}(D, \mathsf{pk}, \bar{k}, \epsilon_{\mathrm{n}})$, to obtain a pertinency vector PV of N ciphertexts, each encrypting 1 or 0. It then proceeds as follows.

Consider first the simple scenario where, throughout the board, there is a single message x_z that is pertinent for the recipient p (but z is initially unknown). Hence, PV has a single $\langle 1 \rangle_{\mathsf{sk}}$ ciphertext, and the rest are $\langle 0 \rangle_{\mathsf{sk}}$.

Let the payload space $\mathcal{P}$ be represented by a tuple of FHE's plaintext space (i.e., for $\mathcal{P} = \{0{,}1\}^{\tilde{n}}$ and the plaintext space $\mathbb{Z}_2$, we use $\tilde{n}$ ciphertexts to represent each payload). In the following we generalize the $\mathsf{FHE.Enc}$, $\mathsf{FHE.Eval}$, and $\langle \cdot \rangle_{\mathsf{sk}}$ notation to work on such tuples in the natural way, as vector operations over $\mathbb{Z}_2$. For each $i \in [N]$, multiply x_i by PV_i homomorphically to get a ciphertext tuple x'_i. Thus, $x'_z \in \langle x_z \rangle_{\mathsf{sk}}$, and $x'_i \in \langle 0 \rangle_{\mathsf{sk}}$ for $i \in [N]$ for the rest of the ciphertext tuples ($i \neq z$). Then, homomorphically sum up all the x'_i to get a ciphertext tuple M', so that $M' \in \langle x_z \rangle_{\mathsf{sk}}$. The detector sends this M' to the recipient, who decrypts it to obtain the desired payload x_z.

Multiple Pertinent Messages via Random Linear Coding. We generalize the above to the case of k pertinent messages, where $0 \leq k \leq \bar{k}$. The cap $\bar{k}$ is chosen by the recipient and given to the detector, but does not need to be known when the board messages are generated.

The above single-message scheme fails if there are multiple pertinent messages, because the detector would output the encrypted *sum* of the k pertinent payloads, from which the individual payloads cannot be (in general) recovered. However, we can have the detector compute several encrypted combinations of the pertinent plaintexts, each one summing them with different weights, in hope of creating a linear system that the recipient can solve.

Specifically, we will choose some $m \geq \bar{k}$ and have the detector homomorphically compute encrypted payload m combinations $\mathsf{Cmb}_j \leftarrow \sum_{i \in [N]} (w_{i,j} \cdot x_i) \cdot \mathsf{PV}_i$ for $j \in [m]$, using random weights $w_{i,j}$ ($i \in [N]$, $j \in m$).[21] Letting $\mathsf{PS} \subseteq [N]$ denote the k pertinent message indices, we then have $\mathsf{Cmb}_j \in \left\langle \sum_{i \in \mathsf{PS}} w_{i,j} \cdot x_i \right\rangle_{\mathsf{sk}}$.

The combinations are realized as FHE ciphertext tuples big enough to represent $\mathcal{P}$, with element-wise operations, i.e., as a vector space over the field $\mathrm{GF}(t)$ of the plaintext space (in our case, of binary plaintext space: $\tilde{n}$ binary ciphertexts, with bitwise AND and XOR).

[21]Here multiplication is in the field $\mathrm{GF}(2)$, for the plaintext space $\mathbb{Z}_2$, so the weights are just 0 or 1. In general, this works over $\mathrm{GF}(t)$ for prime t. From this point on we will require both multiplications and addition (to perform linear algebra), and thus consider the field $\mathrm{GF}(t)$ instead of the group $\mathbb{Z}_t$.

The OMR digest includes the aforementioned output of $\mathsf{OMDt1.Detect}$ of Algorithm 2, from which the recipient can recover PS. Moreover the recipient can know the weights $w_{i,j}$ (by including the seed used to pseudorandomly generate the weights in the digest). Then the recipient can decrypt the payload combinations and recover m equations over the variables $(x_i)_{i\in\mathsf{PS}}$, of the form

$$\sum_{i\in\mathsf{PS}} w_{i,j}x_i = \mathsf{FHE.Dec}(\mathsf{Cmb}_j) \in \Big\langle \sum_{i\in\mathsf{PS}} w_{i,j}\cdot x_i \Big\rangle_{\mathsf{sk}} \quad \text{for } j\in[\bar{k}] \ .$$

If $|\mathsf{PS}| = k$ of these equations are linearly independent, then the recipient can use Gaussian elimination to recover these x_i, i.e., the pertinent messages' payloads.

For randomly-chosen weights, there is a chance that the system is underdetermined (rank less than k). To make this unlikely, it suffices to choose m slightly larger than $\bar{k}$. Working in the field $\mathrm{GF}(t)$ (in our case $t = 2$), and drawing weights uniformly, the probability of getting $\bar{k}$ linearly independent equations is $\prod_{i=m-\bar{k}+1}^{m}(1-1/t^i)$, via [SGGC14, Lemma 1]. Therefore, $m = \bar{k} + \lceil \log_t(\frac{1}{\epsilon_{\mathrm{n}}})\rceil$ suffices. Concretely, $m = \bar{k}+50$ for $t = 2$ gives failure probability $\leq 2^{-50}$.

6.3 Improved Retrieval Using Sparse Random Linear Coding

In the above attempt, the detector's computational cost for preparing the payload combinations is proportional to $\bar{k}\cdot N$. To reduce this cost, we use *Sparse Random Linear Coding (SRLC)* [KS07], encoded homomorphically. SRLC is widely studied and applied to data transmission [KS07, KC16, BJT18, TCL16, GLA17, LMT11].[22]

In this approach, we create weighted linear combinations as above, but we make them sparse: each message is assigned to just a few combinations (i.e., most of the weights $w_{i,j}$ are chosen as zero and the corresponding homomorphic multiplications and additions are thus skipped).

Concretely, for each message index i, the detector pseudorandomly draws a small set $\mathsf{Asg}_i \subset [m]$ of expected size $o(\bar{k}\cdot\log(\epsilon_{\mathrm{n}}^{-1}))$, designating the combinations to which x_i is assigned. It then homomorphically adds $x_i' = x_i \cdot \mathsf{PV}_i$ to Cmb_j for $j \in \mathsf{Asg}_i$.[23]

We then proceed as above in Section 6.2. The resulting Cmb vector is returned to the recipient, together with the message from $\mathsf{OMDt2.Detect}(\cdot)$ from Algorithm 2 and the seed used for the pseudorandom choices. The recipient recovers PS, hence the combination assignments, and by decrypting Cmb it gets a linear system of m (this time, sparse) equations in k variables.

There remains to ensure that this sparse system has full rank k. This requires a suitable choice of m and distribution of Asg,[24] which turns out to be achievable with excellent parameters but difficult to analyze. We thus encapsulate these choices into the notion of SRLC scheme, introduced next.

6.3.1 Defining and Constructing SRLC

Tightly analyzing SRLC parameters is a longstanding, much-studied open problem [KS07, KC16, BJT18]. We proceed to define a notion of SRLC scheme, which can be used as a black box in our construction, and then specify two concrete schemes. $\mathsf{SRLC1}$ has clean analysis using known

[22] State-of-the-art batch-PIR [ACLS18, ALP+21] uses different coding techniques, which rely on the client knowing the pertinent indices a priori.

[23] In general, for plaintext space $\mathrm{GF}(t)$, for $w_{i,j}$ ($j \in \mathsf{Asg}_i$) we use a nonzero weight $w_{i,j}$ that is pseudorandomly drawn from $\mathrm{GF}(t)\setminus\{0\}$. Here, $t = 2$ so this is always 1.

[24] And for $t > 2$, of the distribution of weights over $\mathrm{GF}(t)\setminus 0$.

bounds, and suffices for our asymptotic results. SRLC2 is simpler, faster, and smaller, but relies on empirical estimation for completeness; it will be used in Sections 7 and 10.

Definition 6.3. An SRLC scheme consists of two algorithms

- $(\mathsf{pp}_{\mathsf{SRLC}}, m) \leftarrow \mathsf{GenParams}(1^\lambda, \kappa, \epsilon_\mathrm{F}, t)$: takes as input a security parameter λ, κ (number of columns), ϵ_F (defective rate), and a prime number t, and outputs an SRLC public parameter $\mathsf{pp}_{\mathsf{SRLC}}$.
- $\{(j, w_j)\} \leftarrow \mathsf{GenWeights}(\mathsf{pp}_{\mathsf{SRLC}})$: takes as input an SRLC public parameter $\mathsf{pp}_{\mathsf{SRLC}}$, and outputs a set of indices and weights $\{(j, w_j)\}$ where $j \in [m], w_j \in \mathbb{Z}_t \setminus \{0\}$, representing a sparse vector of length m.

that satisfy the following:

- (Completeness) For any $\kappa \in \mathbb{Z}^+$, and $0 < \epsilon_\mathrm{F} < 1$, let $(\mathsf{pp}_{\mathsf{SRLC}}, m) \leftarrow \mathsf{GenParams}(1^\lambda, \kappa, \epsilon_\mathrm{F}, t)$, and $(S_i \leftarrow \mathsf{GenWeights}(\mathsf{pp}_{\mathsf{SRLC}}))_{i\in[\kappa]}$. Then the matrix $A \in \mathbb{Z}_t^{m\times\kappa}$ defined by

$$A_{i,j} = \begin{cases} w_j & \text{if } (j, w_j) \in S_i \\ 0 & \text{otherwise} \end{cases}$$

fulfills (over the randomness of the algorithms):

$$\Pr[\mathsf{rank}(A) = \kappa] \geq 1 - \epsilon_\mathrm{F} - \mathsf{negl}(\lambda) \ .$$

We construct two different SRLC algorithms as follows. The first SRLC algorithm SRLC1, given in Algorithm 3, is meant for ease of analysis. GenWeights outputs set of indices and weights such that the resulting matrix has independently-drawn entries, with a density of nonzeros set by GenParams.

Algorithm 3 SRLC1: Analytically-Bounded SRLC

```
1: procedure SRLC1.GenParams($1^\lambda, \kappa, \epsilon_\mathrm{F}, t$)
2:     Find the smallest $m$ such that $1 - \prod_{i=1}^{\kappa}(1 - (1 - \frac{3}{\kappa})^{m-i+1}) \leq \epsilon_\mathrm{F}$ and $m \geq 6$
3:     return $(m, \mathsf{pp}_{\mathsf{SRLC}} = (\gamma = 3m/\kappa, \kappa, \epsilon_\mathrm{F}, t, \lambda))$    ▷ $\gamma$ is the expected number of nonzeros
4: procedure SRLC1.GenWeights($\mathsf{pp}_{\mathsf{SRLC}} = (\gamma = 3m/\kappa, \kappa, \epsilon_\mathrm{F}, t, \lambda)$)
5:     $\gamma' \leftarrow \mathsf{B}(m, \gamma/m)$    ▷ binomially-distributed number of nonzeros
6:     $S \leftarrow \{\}, \quad J \leftarrow \{\}$
7:     for $i = 1$ to $\gamma'$ do
8:         $j \xleftarrow{\$} [m] \setminus J$    ▷ draw a new index
9:         $w_j \xleftarrow{\$} \mathbb{Z}_t \setminus \{0\}$    ▷ draw a nonzero weight
10:        $S \leftarrow S \cup \{(j, w_j)\}$
11:    return $S$
```

Lemma 6.4. SRLC1 *in Algorithm 3 is an SRLC scheme for any κ and ϵ_F.*

Algorithm 4 SRLC2: Empirically Bounded SRLC

```
procedure TestRank($\gamma$,$m$,$\lambda$,$\epsilon_\mathrm{F}$,$\kappa$,$t$)
    Ctr $\leftarrow 0$
    for $i = 1$ to $\log(\lambda)\log\log(\lambda)\epsilon_\mathrm{F}^{-1}$ do
        Sample $(S_i)_{i\in[\kappa]}$ where $S_i \leftarrow$ SRLC2.GenWeights($\gamma$, $\mathsf{pp}_\mathsf{SRLC} = (\gamma, m, \epsilon_\mathrm{F}, t, \lambda)$)
        Let $A \in \mathbb{Z}_t^{m\times\kappa}$ be
            $A_{i,j} = \begin{cases} w_j & \text{if } (j, w_j) \in S_i \\ 0 & \text{otherwise} \end{cases}$
        If rank($A$) $< \kappa$: return False
    return True
procedure SRLC2.GenParams($1^\lambda$, $\kappa$, $\epsilon_\mathrm{F}$, $t$)
    $\gamma \leftarrow 3$
    while True do
        for $m = \kappa, \ldots, \kappa \cdot \gamma/2$ do
            if TestRank($\gamma$, $m$,$\lambda$,$\epsilon_\mathrm{F}$,$\kappa$,$t$) = True then
                return ($m$, $\mathsf{pp}_\mathsf{SRLC} = (\gamma, \kappa, \epsilon_\mathrm{F}, t, \lambda)$)
        $\gamma \leftarrow \gamma + 1$
procedure SRLC2.GenWeights($\mathsf{pp}_\mathsf{SRLC} = (\gamma, \kappa, \epsilon_\mathrm{F}, t, \lambda)$)
    $S \leftarrow \{\}$,    $J \leftarrow \{\}$
    for $i = 1$ to $\gamma$ do
        $j \xleftarrow{\$} [m] \setminus J$                                   ▷ draw a new index
        $w_j \xleftarrow{\$} \mathbb{Z}_t \setminus \{0\}$                           ▷ draw a nonzero weight
        $S \leftarrow S \cup \{(j, w_j)\}$
    return $S$
```

Proof. A matrix $A \in \mathbb{Z}_t^{m\times\kappa}$, constructed as in Definition 6.3 using SRLC1, has i.i.d. entries that are nonzero with probability γ/m, and uniform when nonzero. The probability that such A has less than full rank κ is upper bounded by $1 - \prod_{i=1}^{\kappa}(1 - \beta^{m-i+1})$ as shown in [KC16, Lemma 1], where $\beta = \max(1 - \gamma/m, \gamma/(m(t-1)))$, and since we fix $\gamma/m = 3/\kappa$, β is always $1 - \gamma/m$ for $m \geq 6$ because $t \geq 2$. Line 2 thus bounds the failure probability by ϵ_F. Therefore, SRLC1 satisfies the completeness requirement of SRLC. ☐

Complexity. SRLC1 generates sparse vectors of length $m = O(\kappa \log^2 \kappa \log(\epsilon_\mathrm{F}))$, as shown by the following lemma.

Lemma 6.5. *The value m in* SRLC1 *chosen by line 2 in Algorithm 3 is $m = O(\kappa \log^2(\kappa)\log(\epsilon_\mathrm{F}^{-1}))$, and the value γ chosen by line 3 in Algorithm 3 is $m = O(\log^2(\kappa)\log(\epsilon_\mathrm{F}^{-1}))$*

Proof. The expression used to choose m can be bounded as follows (using $0 < \epsilon_\mathrm{F} < 1$ and $1 < \kappa$):

$$1 - \prod_{i=1}^{\kappa}(1 - (1 - \frac{3}{\kappa})^{m-i+1}) \leq 1 - \prod_{i=1}^{\kappa}(1 - (1 - \frac{3}{\kappa})^{m-\kappa+1})$$

$$\leq 1 - (1 - \kappa(1 - \frac{3}{\kappa})^{m-\kappa+1}) \quad = 1 - (1 - (1 - \frac{3}{\kappa})^{m-\kappa+1})^{\kappa} \quad \leq \kappa(1 - \frac{3}{\kappa})^{m-\kappa}$$

Therefore, $\kappa(1-\frac{3}{\kappa})^{m-\kappa} \leq \epsilon$ suffices for the expression in 2 to be fulfilled. Furthermore:

$$\begin{aligned}
&\kappa(1-\frac{3}{\kappa})^{m-\kappa} \leq \epsilon_{\mathrm{F}} && \Leftrightarrow \log(\kappa(1-\frac{3}{\kappa})^{m-\kappa}) \leq \log(\epsilon_{\mathrm{F}}) \\
\Leftrightarrow\ &\log(\kappa)+(m-\kappa)\log(1-\frac{3}{\kappa}) \leq \log(\epsilon_{\mathrm{F}}) && \Leftrightarrow (m-\kappa)\log(1-\frac{3}{\kappa}) \leq \log(\epsilon_{\mathrm{F}}/\kappa) \\
\Leftrightarrow\ & m-\kappa \geq \frac{\log(\epsilon_{\mathrm{F}}/\kappa)}{\log(1-\frac{3}{\kappa})} && \Leftrightarrow m \geq \frac{\log(\epsilon_{\mathrm{F}}/\kappa)}{\log(1-\frac{3}{\kappa})}+\kappa = \frac{\log(\epsilon^{-1}\kappa)}{-\log(1-\frac{3}{\kappa})}+\kappa
\end{aligned}$$

Since $\frac{\log(\epsilon_{\mathrm{F}}^{-1}\kappa)}{-\log(1-\frac{3}{\kappa})} = O(\kappa\log^2(\kappa)\log(\epsilon_{\mathrm{F}}^{-1}))$ and we choose the smallest m in line 2, we indeed get $m = O(\kappa\log^2(\kappa)\log(\epsilon_{\mathrm{F}}^{-1}))$. By line 3, $\gamma = 3m/\kappa = O(\log^2(\kappa)\log(\epsilon_{\mathrm{F}}^{-1}))$. □

The second SRLC algorithm SRLC2, given in Algorithm 4, has simpler GenWeights which just outputs a *fixed* number γ of nonzero elements (with uniformly-random nonzero weight each). However, its completeness relies γ and m being chosen by an empirical estimate (encapsulated within GenParams), since adequately tight analytical bounds are not known.

Lemma 6.6. SRLC2 *in Algorithm 4 is an SRLC scheme for any* κ *and* ϵ_{F}*.*

Proof. Let $p_{\gamma,m}$ denote the probability that a matrix generated as in Definition 6.3 is not full-rank, for given γ,m. For SRLC completeness, it suffices to prove that if $p_{\gamma,m} > \epsilon_{\mathrm{F}}$, then with probability $1-\mathsf{negl}(\lambda)$, TestRank will output False and thus GenParams will not use output these γ, m.

The probability of passing TestRank is the probability that $\epsilon_{\mathrm{F}}\log(\lambda)\log\log(\lambda)$ independent $p_{\gamma,m}$-biased Bernoulli tests are 0. Thus, if $p_{\gamma,m} > \epsilon_{\mathrm{F}}$ then the probability of passing is negligible: $(1-p_{\gamma,m})^n < (1-\epsilon_{\mathrm{F}})^n = ((1-\epsilon_{\mathrm{F}})^{\epsilon_{\mathrm{F}}^{-1}})^{\log(\lambda)\log\log(\lambda)} = O(e^{-\log(\lambda)\log\log(\lambda)}) = O(\lambda^{-\log\log(\lambda)}) = \mathsf{negl}(\lambda)$.

It follows that the output of GenParams fulfills $p_{\gamma,m} \leq \epsilon_{\mathrm{F}} + \mathsf{negl}(\lambda)$.[25] □

6.3.2 Compact OMR using SRLC

We proceed to fully specify an OMR scheme using SRLC, following the intuition of Section 6.2.

Theorem 6.7. *The scheme* OMRt1 *in Algorithm 5 is a Oblivious Message Retrieval scheme, when instantiated with any fully homomorphic encryption scheme* FHE*, PRFs* f*, and SRLC scheme* SRLC*. Moreover, when instantiated with* SRLC1*,* OMRt1 *is also compact.*

Proof sketch . Soundness: Soundness follows from that of Algorithm 2, and returning overflow if the linear system is not fully determined.

Completeness: As in Theorem 6.2, assume for simplicity that there are no false positives.

Completeness of Algorithm 1 ensures that all pertinent messages x_i have $\mathsf{PV}_i \in \langle 1\rangle_{\mathsf{sk}}$. The completeness in Theorem 6.2 ensures that LHS has all the pertinent messages as variables with probability $> 1-\epsilon_{\mathrm{n}}/2-\mathsf{negl}(\lambda)$, given $k \leq \bar{k}$ (otherwise we output overflow and fulfill the requirement). The linear system will be consistent with the correct payloads, unless the FHE decryption errors occur, which happens with negligible probability.

[25] Assuming GenParams runs in time $\mathsf{poly}(\lambda)$. We do not prove this directly, but GenParams.SRLC2 will be empirically executed honestly, and the security model assumes that it is infeasible even for the adversary to run for longer than $\mathsf{poly}(\lambda)$).

Algorithm 5 OMRt1: Oblivious Message Retrieval from generic FHE

```
1: Procedures OMRt1.GenParams, OMRt1.KeyGen and OMRt1.GenClue are identical to OMDt1.
2: procedure OMRt1.Retrieve($\mathsf{BB} = ((x_i, c_i)_{i\in[N]}), \mathsf{pk}_{\mathsf{detect}}, \bar{k}$)
3:                                      ▷ Implicitly taking $\mathsf{pp} = (\ell, \epsilon_{\mathrm{p}}, \epsilon_{\mathrm{n}}, \mathsf{pp}_{\mathsf{FHE}})$
4:     Draw a random seed $s$
5:     $\hat{k} \leftarrow \bar{k} + N\log(N)\epsilon_{\mathrm{p}}$
6:     $(\mathsf{pp}_{\mathsf{SRLC}}, m) \leftarrow \mathsf{SRLC1.GenParams}(1^\lambda, \hat{k}, \epsilon_{\mathrm{n}}/2, 2)$
7:     $\mathsf{PV} \leftarrow \mathsf{OMDt1.Detect}(D, \mathsf{pk}, \bar{k}, \epsilon_{\mathrm{n}}/2)$ (from Algorithm 1)
8:     Initialize combinations $(\mathsf{Cmb}_i = (\mathsf{FHE.Enc}(\mathsf{pk}_{\mathsf{detect}}, 0^{\tilde{n}})))_{i\in[m]}$
9:     for $i = 1$ to $N$ do
10:        $S \leftarrow \mathsf{SRLC1.GenWeights}(\mathsf{pp}_{\mathsf{SRLC}})$
11:        for $(j, w_j) \in S$ do
12:            $\mathsf{tmp} \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}_{\mathsf{detect}}, \mathsf{tmp}, w_j, \times)$
13:            $\mathsf{Cmb}_j \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}_{\mathsf{detect}}, \mathsf{tmp}, \mathsf{Cmb}_j, +)$
14:    $\mathsf{CLHS} \leftarrow \mathsf{OMDt2.Detect}(D, \mathsf{pk}, \bar{k}, \epsilon_{\mathrm{n}}/2)$
15:    return $M = (s, \mathsf{CLHS}, \mathsf{CRHS} = (\mathsf{Cmb}_i)_{i\in[m]}, \mathsf{pp}_{\mathsf{SRLC}})$
16: procedure OMRt1.Decode($M$, sk)     ▷ Implicitly taking $\mathsf{pp} = (\ell, \epsilon_{\mathrm{p}}, \epsilon_{\mathrm{n}}, \mathsf{pp}_{\mathsf{FHE}}, \mathsf{pp}_{\mathsf{SRLC}})$
17:    Parse $(s, \mathsf{CLHS}, \mathsf{CRHS}, \mathsf{pp}_{\mathsf{SRLC}}) \leftarrow M$
18:    $\mathsf{PL}_{\mathsf{lhs}} \leftarrow \mathsf{OMDt2.Decode}(\mathsf{CLHS}, \mathsf{sk})$
19:    if $\mathsf{PL}_{\mathsf{lhs}} = \mathsf{overflow}$ then
20:        return overflow
21:    for $i = 1$ to $m$ do
22:        $\mathsf{LHS}_i \leftarrow 0$   ▷ As a linear combination in formal variables $x_i$, updated iteratively below
23:    for $i \in \mathsf{PL}_{\mathsf{lhs}}$ do
24:        $S \leftarrow \mathsf{SRLC1.GenWeights}(\mathsf{pp}_{\mathsf{SRLC}}; s)$
25:        for $(j, w_j) \in S$ do
26:            $\mathsf{LHS}_j \leftarrow \mathsf{LHS}_j + w_j \cdot x_i$
27:    $\mathsf{RHS} \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, \mathsf{CRHS})$
28:    return the result of solving LHS = RHS by Gaussian elimination, or overflow if failed
```

Moreover, the LHS of the linear system needs to be full rank. This is implied, with all but negligible probability, by completeness of SRLC1 (Lemma 6.4). Note that although we have invoked GenWeights for each of the N messages in the board, at most $\hat{k}$ of these (i.e., the messages corresponding to messages detected as pertinent, including false positives) contribute columns to the matrix. Therefore, the probability that the linear system is not full rank is bounded by $\epsilon_{\mathrm{n}}/2$. Note that SRLC completeness is defined for $\kappa = \hat{k}$, but it trivially follows that matrices generated by $\kappa' < \kappa$ invocations of GenWeights are also of full rank κ' with at least the same probability. Note that the completeness of SRLC is unaffected by adversarial input, so its average-case guarantee suffices.

Altogether, for $k \leq \bar{k}$, the failure probability is $\leq \epsilon_{\mathrm{n}} + \mathsf{negl}(\lambda)$.

Privacy: Privacy directly follows from proof for Algorithm 2. There is no additional information placed in the board or given to the detector.

Compactness: When OMRt1 is instantiated with SRLC1, the digest size is $O(\hat{k}\log(\hat{k})\log(\epsilon_{\mathrm{n}}^{-1})\log(N))$.

By Lemma 6.5, we will get some $m = O(\hat{k}\log^2(\hat{k})\log(\epsilon_{\text{n}}^{-1}))$ combinations for retrieval, each with the size of payload number of ciphertexts. Since we consider the payload size to be constant, and $\hat{k} \leq N$, the cost is dominated by $O(\hat{k}\log(\hat{k})\log(\epsilon_{\text{n}}^{-1})\log(N))$ from Algorithm 2 for detecting pertinent message indices. □

Computational Complexity. We analyze OMRt1 complexity when instantiated SRLC1. The computational complexity of the detector is $O(N(\log(\hat{k})\log(\epsilon_{\text{n}}^{-1})\log(N) + \log(1/\epsilon_{\text{p}})))$, i.e., quasi-linear in N and sublinear in $\bar{k}$. This is since $\gamma = O(\hat{k}\log^2(\hat{k})\log(\epsilon_{\text{n}}^{-1})$ by Lemma 6.5 and therefore dominated by computational cost in Algorithm 2, similar as compactness. As analyzed above for communication, the cost is dominated by the cost of detection as in Algorithm 2, as $\gamma = O(\log^2(\hat{k})\log(\epsilon_{\text{n}}^{-1})) = O(m/\hat{k})$. The recipient's computation complexity is $O(\hat{k}\log(\hat{k})\log(\epsilon_{\text{n}}^{-1})\log(N) + \hat{k}^3)$, for the digest decryption and Gaussian elimination respectively. Note that the computation and communication complexity are both independent of adversarial input (except for the board size).

Alternative Construction. An alternative construction for OMR is deferred to Appendix B.3, since it does not extend a practical construction. Its main idea is the same as the free-bucket-search alternative mentioned in the last paragraph of Section 6.1.2.

(Im)practicality. The above establishes the asymptotic existence of compact OMR (assuming existence of FHE), but it is still impractical, e.g., due to the cost of the Recrypt algorithm in state-of-the-art FHE schemes [DM15, CGGI20, Bra12, FV12, BGV12]. Moreover, the clues are large (e.g, BFV has a minimum ciphertext size of roughly 30 kB for typical ring dimensions like 2^{15} [PAL21, Mic20], and FHEW/TFHE needs 512 bytes per each of the ∼20 plaintext bits required in Section 6.1.1).

7 Practical OMR

We proceed to introduce optimizations that improve communication and computation costs to practical levels, as well as security improvements. See Section 3.2 above for an overview, and Section 10 below for implementation and quantitative evaluation.

Our starting point is the generic FHE construction of Section 6, generalized from plaintext space $\mathbb{Z}_2$ to $\mathbb{Z}_t$ for prime $t \geq 2$ (as discussed in footnotes whenever pertinent). The generalization immediately gives us improved concrete bounds on false positive rate, and in the SRLC-based retrieval. (Moreover, it will match the native arithmetic operations of BFV homomorphic operations discussed below.) We then proceed to modify the scheme as follows.

7.1 PVW Clue Ciphertext

Instead of generating clues using encryption under an FHE scheme, we use a lighter-weight encryption scheme which can still be homomorphically decrypted and processed by the detector. Specifically, we use the PVW scheme [PVW08], which is a variant of the Regev05 LWE-based encryption scheme [Reg09]. See Section 5.2 for details. The PVW decryption algorithm can be cheaply evaluated under FHE, and moreover its ciphertext size grows slowly when multiple bits are encrypted. Choice of PVW parameters, such as lattice dimension n and noise level σ, is discussed below in Sections 7.3 and 10.1.

The switch to PVW maintains the OMD/OMR privacy requirement, since PVW is IND–CPA and key-private: both the ciphertexts and public keys are indistinguishable, even taken together, from uniformly-drawn field elements [Hal05][PVW08, Lemma 7.4].

FHE-Friendly Decryption. PVW.Dec can be divided into three parts: an inner product, a division, and a rounding into $\mathbb{Z}_2$ as shown in Section 5.2. All these parts can be done homomorphically with reasonable cost using FHE. We discuss in more detail later how we perform these operations.[26]

Reduced Clue and Key Size. Using PVW, the clue size is $O(n + \ell)$ for the ℓ ciphertexts (compared to $O(n \cdot \ell)$ in the original Regev05 scheme [Reg09]). Moreover, to reduce the public key size, instead of explicitly including the random matrix $\mathbf{A}$ in the clue key, we generate the matrix pseudorandomly from a random seed s_A and inclue s_A in the clue key. This does not affect security or correctness [Reg09].

7.2 BFV Leveled Homomorphic Encryption

In the detection and retrieval algorithm, we also replace the general FHE with leveled HE, i.e., homomorphic encryption restricted to evaluation of arithmetic circuits with predetermined multiplicative depth. This suffices since, as shown below, the multiplicative depth can be kept low and moreover, after the switch to PVW encryption, we do not need the Recrypt functionality (which is expensive and increases the scheme's overall parameters). Specifically, we use the BFV [Bra12, FV12] leveled HE scheme (see Section 5.3.1).

The recipient now also generates a BFV key pair, and adds the BFV public key to the detection key, which now contains the BFV public key, and the PVW secret key PVW.sk encrypted under BFV public key. The detector uses these to homomorphically decrypt the PVW ciphertexts in the clue, resulting in PV_i which are $\langle 0 \rangle_{\mathsf{sk}}^{\mathsf{BFV}}$ or $\langle 1 \rangle_{\mathsf{sk}}^{\mathsf{BFV}}$. It then proceed to process these into a digest as in Section 6, using BFV homomorphic operations, all with plaintext space GF(t).

One advantage of this choice is that BFV supports packing and SIMD operations: each BFV ciphertext has D plaintext *slots*, each of which can convey an element of $\mathbb{Z}_t$, for some plaintext modulus t and a parameter D; we can compute over these D elements simultaenously the same homomorphic operation (see Section 5.3.1).

When computing PV, we can thus operate on D separate messages $\{(x_i,c_i)\}$ at a time, in parallel via SIMD. For simplicity, consider first the $\ell = 1$ case, i.e., single bit encrypted in each clue (we later add amplify soundness analogously to Section 6.1.1.) We take D PVW ciphertexts at a time (each containing $n + 1$ elements of $\mathbb{Z}_t$) and perform D-parallel SIMD homomorphic computation on these, to essentially attain a packed version of OMDt1.Detect, i.e., produce one PV ciphertext whose i-th slot encrypts 1 if the i-th message among the D is pertinent, or 0 otherwise. For $\ell > 1$, we can repeat this procedure with ℓ different PVW secret keys encrypted as BFV ciphertexts.

Specifically, the detector performs the following steps to homomorphically perform PVW.Dec (described intuitively here, and precisely in Algorithm 6). Note that for simplicity when decrypting homomorphically using BFV, we redefine our clues to be PVW encryptions of 0 instead of 1 as above (i.e. $\mathsf{PVW.Dec}(\mathsf{sk}_{p'}, \mathsf{GenClue}(\mathsf{pk}_p, \cdot)) = 0$ iff $p = p'$ w.h.p.).

- **Inner Product (InnerProd).** The detector performs the first step of PVW.Dec: inner product of the clue's PVW ciphertext with PVW.sk (that is provided by the recipient under BFV

[26]Other asymmetric encryption schemes would cause heavier FHE computation for the detector, but may offer smaller clue sizes.

encryption). This homomorphic operation is linear (the clue is known and thus handled as a constant), and thus cheap.

- **Range checking (RangeCheck)** For a range $[-r, r]$ and plaintext element $u \in \mathbb{Z}_t$, this maps u to 0 if $u \in [-r, r]$, otherwise to 1. We implement this homomorphically using [INZ21, Equation 2] as follows, using the Paterson-Stockmeyer algorithm [PS73] to minimize multiplicative depth. To evaluate the function

$$f(x) \quad = \quad \begin{cases} 1 & \text{if } t-r \le x \le r \\ 0 & \text{otherwise} \end{cases}$$

we can evaluate the following polynomial over $\mathbb{Z}_t$:

$$\mathsf{RC}_r(X) = f_r(0) - \sum_{i=0}^{t-1} X^i \sum_{a=0}^{t-1} f_r(a) a^{t-1-i} \quad . \tag{1}$$

We thus calculate $u' \leftarrow \mathsf{RC}_r(u)$ homomorphically, so $u' \in \langle 1 \rangle_{\mathsf{sk}}^{\mathsf{BFV}}$ iff $u \in [-r, r]$ (which is the case for pertinent messages with high probability). An exact implementation of PVW.Dec would require $r = t/4$ (by definition of PVW in Section 5.2), but this can be relaxed to reduce the false postiive rate. Since the error distribution is a Gaussian with standard deviation $\sqrt{w}\sigma$, the resulting decryption failure probability (and thus false-negative rate) is $\le \mathsf{erf}(r/\sqrt{2w}\sigma)$. We can bound this false negative rate at, e.g., $< 2^{-40}$ by setting $r = 7.2\sqrt{w}\sigma$.

- **PV Unpacking (PVUnpack).** Because we used SIMD evaluation, the above steps result in a single ciphertext ct whose D slots encode $u' \in \{0,1\}$ values of D different messages. This already suffices as a digest (decrypting it lets the recipient find out which messages are pertinent), but to maintain a clean interface and allow further improvements below, we proceed to unpack ct's slots into separate D ciphertexts $(\mathsf{PV}_i)_{i \in [D]}$. This is done, for each $i \in [D]$, by multiplying ct by I_i (where I_i is a plaintext vector 1 in slot i and 0 in the rest), and then spreading the i-th slot to all slots by repeatedly rotating and summing (see [HHCP18]).

Between RangeCheck and PVUnpack, we flip the sign of u' ($u' \leftarrow 1 - u'$) so that after all these steps, we get PV_i ciphertexts encrypting 1 in all slots for the pertinent messages, and 0 in all slots for impertinent messages with some false positive rate. Similarly to Section 6, we proceed as follows.

Soundness Amplification. The above incurs false positives when decryption of impertinent messages happens to result in $0 \in \mathbb{Z}_t$. Analogously to in Section 6.1.1, we reduce this false positive rate by having the sender encrypt ℓ 0's to the recipient's clue key (in this case: all in a single ℓ-bit PVW ciphertext); and having the detector check homomorphically whether all ℓ resulting inner products fall within the range $[-r,r]$, by multiplying the ℓ outputs of range checking. Note that here, the probability of a clue ciphertext decrypting to 0 (so that the range-checked output is $\mathsf{PV} \in \langle 1 \rangle_{\mathsf{sk}}^{\mathsf{BFV}}$) for impertinent messages is $p = 1 - (2r+1)/q \ge 1/2$, rather than 1/2 as in the $\mathbb{Z}_2$ case.[27] Therefore, using ℓ ciphertexts, we get ϵ_{p} of p^ℓ.[28]

[27]This holds because that for an honestly generated PVW secret key sk, the matrix multiplication between sk and PVW ciphertexts encrypted under a different secret key gives a distribution that is statistically close to an uniformly random distribution in $\mathbb{Z}_q^\ell$.

[28]This decision predicate (AND of the range check on all ℓ inner products) is suboptimal. The error rate could be

Algorithm 6 Homomorphic Decryption Auxiliary Functions

```
1: procedure InnerProd($\mathsf{pp_{BFV}}, \mathsf{pk_{BFV}}, \mathsf{ct_{pvwSK}}, \mathsf{pln} = (\mathsf{pln}_i)_{i\in[n+\ell]}$)
2:     Parse $\mathsf{ct_{pvwSK}} = (\mathsf{ct}_j)_{j\in[\ell]} = ((\mathsf{ct}_1, \dots, ct_n)_j)_{j\in[\ell]}$
3:     for $i = 1$ to $\ell$ do
4:         $\mathsf{res}_i = \mathsf{BFV.Eval}(\mathsf{pk_{BFV}}, \mathsf{ct}_i, (\mathsf{pln}_j)_{j\in[n]}, \text{inner product})$
5:         $\mathsf{res}_i = \mathsf{BFV.Eval}(\mathsf{pk_{BFV}}, \mathsf{pln}_{n+i}, \mathsf{res}_i, \mathsf{Sub})$
6:     return $\mathsf{res} = (\mathsf{res}_i)_{i\in[\ell]}$
7: procedure RangeCheck($\mathsf{pk_{BFV}}, \mathsf{ct} = (\mathsf{ct}_i)_{i\in[\ell]}, r$)
8:     Parse $\mathsf{ct} = (ct_i)_{i\in[\ell]}$
9:     for $i = 1$ to $\ell$ do
10:        $\mathsf{res}_i \leftarrow \mathsf{BFV.Eval}(\mathsf{pk_{BFV}}, \mathsf{ct}_i, \mathsf{RC})$          ▷ RC as defined in Eq. (1)
11:    return $\mathsf{res} = (res_i)_{i\in[\ell]}$
12: procedure PVUnpack($\mathsf{pp_{BFV}}, \mathsf{pk_{BFV}}, \mathsf{ct}$)
13:    for $i = 1$ to $D$ do
14:        $\mathsf{tmp} = (0, \dots, 1, \dots, 0)$ where $\mathsf{tmp}_i = 1$
15:        $\mathsf{res}_i = \mathsf{BFV.Eval}(\mathsf{pk_{BFV}}, \mathsf{ct}, \mathsf{tmp}, \times)$
16:        for $j = 1$ to $\log(D)$ do
17:            $\mathsf{ct_{tmp}} \leftarrow \mathsf{BFV.Eval}(\mathsf{pk_{BFV}}, \mathsf{res}_i, \mathsf{Rotate}(j))$
18:                          ▷ Rotate($j$) means rotating the plaintext vector to right by $j$ slots.
19:            $\mathsf{res}_i \leftarrow \mathsf{BFV.Eval}(\mathsf{pk_{BFV}}, \mathsf{ct_{tmp}}, +)$
20:    return $\mathsf{res} = (\mathsf{res}_i)_{i\in[D]}$
```

Applying all of the above, we obtain and OMD scheme analogous to Algorithm 1, but based on PVW clues and BFV scheme instead of generic FHE, thereby improving efficiency and size. The same technique applies to Algorithm 2, for more efficient *compact* detection.

Finally, compress the detection digest by one of two means. (The full resulting algorithms appear later in this section.)

Deterministic Digest Compression. Each BFV ciphertext can pack D elements of $\mathbb{Z}_t$, each of which can represent $\lfloor \log(t) \rfloor$ bits of plaintext information. We pack the single-bit pertinency indicators into these bits, homomorphically (starting with the above representation, where each bit occupies a whole slot in the digest, or a whole ciphertext after PV unpacking). To achieve this dense packing, for the i^{th} message we homomorphically add 2^ℓ to slot j, where $j \cdot \lfloor \log(t) \rfloor + \ell = i$ and $j \in [D], \ell \in [\lfloor \log(t) \rfloor]$, given $D\lfloor \log(t) \rfloor \geq N$. This naturally extends to $N > D\lfloor \log(t) \rfloor$ by adding more ciphertexts. This makes near-optimal use of the plaintext space; and in terms of the ciphertext size (and thus digest size), utilization remains high: roughly **4.5 bits per message**, for the representative parameters of Section 10.

The recipient can sum up all these bits to get the exact number of pertinent messages, and thus robustly detects overflow.

reduced by Bayesian deduction on the inner products before range check. Specifically: compute the log-likelihood of each ciphertext's inner product being observed for a true positive vs. negative; sum these log-likelihoods; and threshold the result. However, this computation (or a useful approximation thereof) would increase multiplicative depth and thus BFV costs.

Randomized Digest Compression. Alternatively, we can use the bucket-based probabilistic method of Section 6.1.2 to achieve an compact digest, and moreover, we can compress D accumulators into one ciphertext to fully utilize all D slots in each ciphertext. In particular, we can achieve an amortized digest size of **less than 1 bit per messsage**, including retrieval, for sufficiently large $N \gg \bar{k}$ (e.g., $N = 10{,}000{,}000$, $\bar{k} = 50$). See details in Algorithm 8 and performance in Section 10.2.

For this method, we need further care for the recipient to precisely count the number of pertinent messages and thus detect overflow. We could use the summation of individual bucket counters to get k, but this may still overflow if the number of pertinent messages is huge, since each slot is computed homomorphically in the plaintext space $\mathbb{Z}_t$ (e.g. if $k \gtrsim tD$, where $tD > 10^{12}$ for typical parameters).

If the above parameters are exceeded, we can still avoid undetected decoding failures by keeping a global counter that can represent values in $[N]$ without overflow. This can be realized using $\lceil \log_t(N) \rceil$ ciphertexts that representing the counter in t-ary, and the correspoding big-integer additions are done via homomorphic evaluation. This is expensive under BFV and not needed in our parameter regime.

OMD via Deterministic Digest Compression. We can use deterministic digest compression to construct a non-compact OMD, which we call $\mathsf{OMDp1}$. The advantage of this method is that it does not need $\mathsf{PVUnpack}$: for the i-th message, $\mathsf{OMDp1}$ encrypts $2^{i/D}$ and multiply with PV_i using SIMD (so perform D of them in parallel), and adds the result together. Therefore, it greatly reduces the detector running time. Since it is a simplified version of $\mathsf{OMRp1}$, we omit the pseudocode. (See Appendix A for a summary of all the schemes introduced in this paper.)

7.3 A Practical OMR scheme

Fig. 2 portrays the high-level components of the resulting scheme, and their invocation of different encryption schemes.

By combining all of the above detection optimizations, and adding message *retrieval* analogously to Section 6.3.2, we have a practical OMR scheme $\mathsf{OMRp1}$, as given in Algorithm 7. Here we use the aforementioned *Deterministic Digest Compression*, which is simpler than compact (randomized) detection, and offers better concrete digest size and detection time for some parameter choices of interest (cf. Section 10).

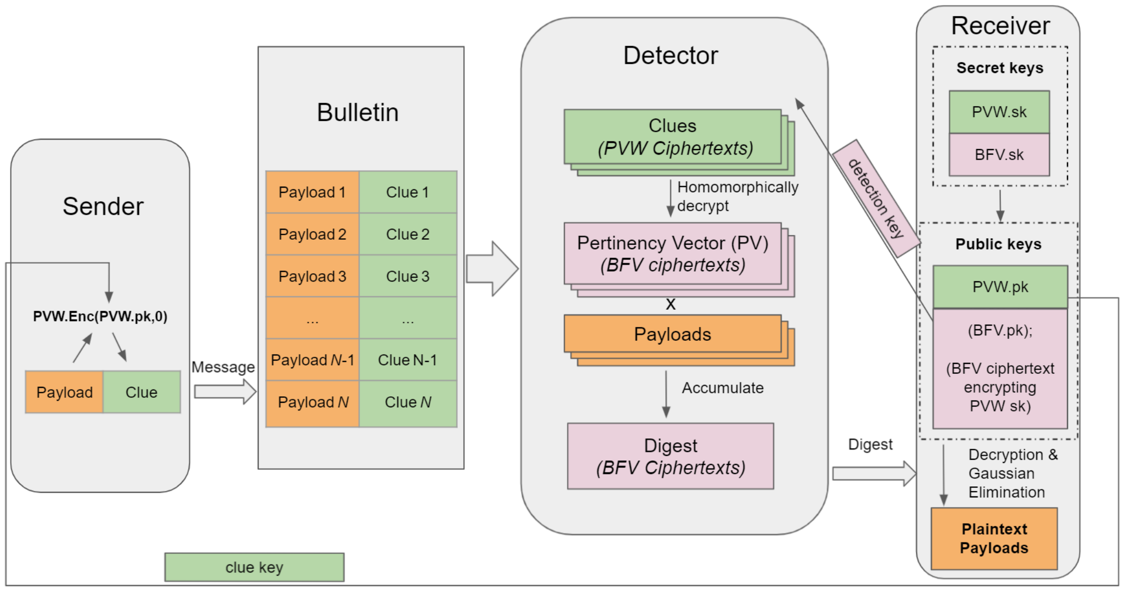


Figure 2: Main components of our scheme based on PVW and BFV.

Setting Parameters. The parameters to decide are BFV parameters D, t (which then imply the remaining parameters in $\mathsf{pp}_{\mathsf{BFV}}$ [ACC+18]), the LWE parameters n, w, ℓ, q, σ (we fix $q = t$), and the range r. The choice should satisfy the following criteria (given the security parameter λ, false positive rate ϵ_{p} and false negative rage ϵ_{n}):

1. PVW encryption using n,q,σ is semantically secure.
2. BFV encryption using t, D is semantic secure.
3. Moreover, this BFV parametrization should allow the homomorphic computation of the circuit described in algorithm OMRp1.Retrieve, with failure probability $\mathsf{negl}(\lambda)$ implied by the correctness of BFV [ACC+18].
4. $\ell \cdot (1 - \mathsf{erf}(r/(\sqrt{2w}\sigma))) < \epsilon_{\mathrm{n}}/2$. Here, $\mathsf{erf}(r/(\sqrt{2w}\sigma)$ is the probability a PVW ciphertext in the pertinent message to be decrypted to 0 (i.e., the inner product is in $[-r, r]$), as it the sum of at most LWE w samples and thus distributed as a Gaussian with standard deviation $\sqrt{w}\sigma$. A false negative happens if any of the ℓ ciphertexts does not decrypt to 0, and we take a union bound over these events.
5. $((2r + 1)/t)^{\ell} < \epsilon_{\mathrm{p}}$, where $(2r + 1)/t$ is the probably that a impertinent clue ciphertext is decrypted into 0 (if this happens for all ℓ ciphertexts in the clue, we get a false positive), and $r \leq t/2$, where the LWE parameter σ depends on t and λ, and w depends on t (see [APS15, Pla18, PVW08]).[29] [APS15, Pla18, PVW08]

[29]In practice, the first item can be instead replaced by $\left(1 - \ell(\mathsf{erf}(r/(\sqrt{2w'}\sigma)) \cdot \mathsf{erf}(\frac{w' - w/2}{\sqrt{2w/12}}))\right) < \epsilon_{\mathrm{n}}/2$ such that $w' < w$ and minimizes r. For the more easily understandable analysis in our paper, we do not use this directly. $\mathsf{erf}(\frac{w' - w/2}{\sqrt{2w/12}})$ should be replaced with 1 if $w' = w$, as its a discrete distribution approximated by a normal distribution.

Algorithm 7 OMRp1: Practical Oblivious Message Retrieval

Let $f_s(x)$ be a PRF. Let BFV and PVW be as defined above.

```
1: procedure OMRp1.GenParams($1^\lambda, \epsilon_{\rm p}, \epsilon_{\rm n}$)
2:     Choose $\mathsf{pp}_{\mathsf{BFV}} = (D, t, \dots)$, $\mathsf{pp}_{\mathsf{PVW}} = (n, w, \ell, q, \sigma)$, and range $r$    ▷ See Setting parameters in §7.3
3:     return $\mathsf{pp} = (1^\lambda, \epsilon_{\rm n}, \epsilon_{\rm p}, \mathsf{pp}_{\mathsf{BFV}}, \mathsf{pp}_{\mathsf{PVW}}, r, \ell)$    ▷ Provided implicitly below
4: procedure OMRp1.KeyGen
5:     $(\mathsf{sk}_{\mathsf{PVW}}, \mathsf{pk}_{\mathsf{PVW}}) \leftarrow \mathsf{PVW.KeyGen}()$
6:     $(\mathsf{sk}_{\mathsf{BFV}}, \mathsf{pk}_{\mathsf{BFV}}) \leftarrow \mathsf{BFV.KeyGen}()$
7:     $\mathsf{ct}_{\mathsf{pvwSK}} \leftarrow \mathsf{BFV.Enc}(\mathsf{pk}_{\mathsf{BFV}}, \mathsf{sk}_{\mathsf{PVW}})$
8:     return $(\mathsf{sk} = (\mathsf{sk}_{\mathsf{BFV}}), \mathsf{pk} = (\mathsf{pk}_{\mathsf{clue}} = \mathsf{pk}_{\mathsf{PVW}}, \mathsf{pk}_{\mathsf{detect}} = (\mathsf{pk}_{\mathsf{BFV}}, \mathsf{ct}_{\mathsf{pvwSK}})))$
9: procedure OMRp1.GenClue($\mathsf{pk}_{\mathsf{clue}}, x$)
10:    $\vec{b} \leftarrow (0, \dots, 0) \in \mathbb{Z}_t^\ell$
11:    $c \leftarrow \mathsf{PVW.Enc}(\mathsf{pk}_{\mathsf{clue}}, \vec{b})$    ▷ Recall: clue $c \in \mathbb{Z}_t^{n\times\ell}$
12:    return $c$
13: procedure OMRp1.Retrieve($\mathsf{BB}, \mathsf{pk}_{\mathsf{detect}}, \bar{k}$)
14:                                                   ▷ Phase 1: Initialization
15:    Draw a random seed $s = (s_f, s_h)$
16:    Parse $\mathsf{BB} = \{(x_1, c_1), \dots, (x_N, c_N)\}$ and $\mathsf{pk}_{\mathsf{detect}} = (\mathsf{pk}_{\mathsf{BFV}}, \mathsf{ct}_{\mathsf{pvwSK}})$
17:    Let $C \leftarrow N/(D \cdot \log(t))$
18:    Initialize $(\mathsf{Acc}_i = \mathsf{BFV.Enc}(\mathsf{pk}_{\mathsf{BFV}}, (0, \dots, 0)))_{i\in[C]}$    ▷ $D$ zeros
19:                                   ▷ Phase 2: detection in batches of $D$ messages
20:    for $i = 1$ to $N/D$ do    ▷ Assume wlog that $D$ divides $N$
21:        Parse each clue as $c_{iD+i'} = ((c_{i',\kappa}))_{\kappa\in[n+\ell]} \in \mathbb{Z}_q^{n+\ell}$ for $i' \in [D]$
22:        Let $\bar{c}_\kappa = (c_{i',\kappa})_{i'\in[D]}$ for $\kappa \in [n+\ell]$    ▷ $\bar{c}_\kappa$ lists the $\kappa$-th element of every PVW clue in this batch
23:        $\alpha_1 \leftarrow \mathsf{InnerProd}(\mathsf{pp}_{\mathsf{BFV}}, \mathsf{pk}_{\mathsf{BFV}}, \mathsf{ct}_{\mathsf{pvwSK}}, (\bar{c}_\kappa)_{\kappa\in[n+\ell]})$
24:        $\alpha_2 \leftarrow \mathsf{RangeCheck}(\mathsf{pp}_{\mathsf{BFV}}, \mathsf{pk}_{\mathsf{BFV}}, \alpha_1, r)$
25:        $\alpha_3 = \prod_{i=0}^{\ell-1}(1 - \alpha_2[i])$
26:                    ▷ For OMDp1, stop here and return all $\alpha_3$'s after finishing the loop.
27:        $(\mathsf{PV}_{i'})_{i'\in((i-1)D, iD]} \leftarrow \mathsf{PVUnpack}(\mathsf{pp}_{\mathsf{BFV}}, \mathsf{pk}_{\mathsf{BFV}}, \alpha_3)$
28:        for $i' = 1$ to $D$ do
29:            $y \leftarrow \lfloor\frac{((i-1)D+i')}{(C\cdot D)}\rfloor$, $z \leftarrow \lfloor\frac{((i-1)D+i') \bmod (C\cdot D)}{D}\rfloor$, $\omega \leftarrow ((i-1)D + i') \bmod t$
30:            $\mathsf{Acc}_y[z] \leftarrow \mathsf{Acc}_y[z] + 2^\omega$    ▷ Homomorphic slot-wise addition
31:                                                   ▷ Phase 3: Finalization
32:    $\hat{k} \leftarrow \bar{k} + N\log(N)\epsilon_{\rm p}$,    $(\mathsf{pp}_{\mathsf{SRLC}}, m) \leftarrow \mathsf{SRLC.GenParams}(1^\lambda, \hat{k}, \epsilon_{\rm n}/2, t)$
33:                ▷ In practice $(\mathsf{pp}_{\mathsf{SRLC}}, m)$ is preprocesssed and tabulated and therefore becomes $O(1)$
34:    Initialize combinations $\{\mathsf{Cmb} = \mathsf{BFV.Enc}(\mathsf{pk}, 0)\}_{k\in[m]}$
35:    for $i = 1$ to $N$ do
36:        $S \leftarrow \mathsf{SRLC.GenWeights}(\mathsf{pp}_{\mathsf{SRLC}})$
37:        for $(j, w_j) \in S$ do
38:            $\mathsf{Cmb}_j = \mathsf{Cmb}_j + \mathsf{PV}_i \cdot x_i \cdot w_j$    ▷ by homomorphic addition and scalar multiplication
39:    return $M = (s, (\mathsf{Acc}_y)_{y\in[C]}, (\mathsf{Cmb}_k)_{k\in[m]}, \mathsf{pp}_{\mathsf{SRLC}})$
40: procedure OMRp1.Decode($M, \mathsf{sk}$)
41:    Parse $M = (s, (\mathsf{Acc}_y)_{y\in[C]}, (\mathsf{Cmb}_k)_{k\in[m]}, \mathsf{pp}_{\mathsf{SRLC}})$    ▷ $C = |(\mathsf{Acc})|, m = |(B)|$
42:    $(\mathsf{tmp}_y)_{y\in[C]} \leftarrow \mathsf{BFV.Dec}(\mathsf{pk}, (\mathsf{Acc}_y)_{y\in[C]})$ where $C = |(\mathsf{Acc})|$ , $k \leftarrow 0$
43:    for $i = 1$ to $C$ do
44:        for $j = 1$ to $D$ do
45:            for $k = 1$ to $\lfloor\log(t)\rfloor$ do
46:                if $\mathsf{tmp}_i[j]$'s $k^{th}$ bit $= 1$ then
47:                    Append $((i-1)D\lfloor\log(t)\rfloor + (j-1)\lfloor\log(t)\rfloor + k)$ to $\mathsf{PL}_{\mathsf{lhs}}$
48:                    $k \leftarrow k + 1$
49:    If $k > \bar{k}$, return overflow
50:    for $i \in \mathsf{PL}_{\mathsf{lhs}}$ do
51:        $S \leftarrow \mathsf{SRLC.GenWeights}(\mathsf{pp}_{\mathsf{SRLC}})$
52:        for $(j, w_j) \in S$ do
53:            $\mathsf{LHS}_j \leftarrow \mathsf{LHS}_j + w_j \cdot x_i$
54:    $\mathsf{RHS} \leftarrow \mathsf{BFV.Dec}(\mathsf{sk}, (\mathsf{Cmb}_k)_{k\in[m]}))$
55:    return the result of solving LHS = RHS by Gaussian elimination, or overflow if failed
```

Note that, for given error parameters, these requirements can always be satisfied using sufficiently large t, since other parameters that depend on t only grows as $o(t)$. Concretely, Section 10.1 proposes a set of parameters satisfying these requirements. Our parameter suggestions also provide practical efficiency as shown in Section 10.2.[30]

Theorem 7.1. *The scheme* OMRp1 *in Algorithm 7 is an* OMR *scheme, assuming security of* PVW *encryption (Section 5.3), security of* BFV *leveled HE (Section 5.3.1), when instantiated with PRF f and an SRLC scheme* SRLC.

Proof sketch . Completeness: Note that the false positive samples has negligible effect as in Theorem 6.2. Therefore, we assumes there is no false positives in our proof for simplicity.

A pertinent message payload x_i is successfully retrieved if:

1. Its clue's PVW ciphertext decrypts correctly using the range $[-r,r]$.

2. The detector's homomorphic evaluation, decrypted by the recipient, does not err. This is implied (up to negligible error) by the correctness of BFV.

3. Gaussian elimination succeeds (the equations are linearly independent)

Note that we use deterministic index retrieval: if conditions 1 and 2 are fulfilled, we can obtain the index i automatically, so the detection does not bring extra failure probability.

Condition 1 holds with probability ϵ_{n} because we have chosen range r to have $(\ell\cdot(1-\mathsf{erf}(\frac{r}{\sqrt{2w}\sigma}))) < \epsilon_{\mathrm{n}}/2$. This is achievable for any $\epsilon_{\mathrm{n}} > \mathsf{negl}(\lambda)$, as $r = t/4$ (i.e. the decryption range being $[-t/4, t/4]$ for zero) gives failure probability of $\mathsf{negl}(\lambda)$ given the correctness of the underlying PVW scheme.

Condition 3 holds with probability $1 - \epsilon_{\mathrm{n}}/2 - \mathsf{negl}(\lambda)$ similarly to Theorem 6.7: by the completeness of SRLC, the linear system has rank k (for $k \leq \bar{k}$); and by the soundness of BFV, it is consistent.

Therefore, the total failure probability is $< \epsilon_{\mathrm{n}} + \mathsf{negl}(\lambda)$ as required.

Soundness: False positives occur when for all ℓ parts of the PVW ciphertext in a clue, their inner products with pvwSK key fall into range $\pm r$. This has probability $((2r+1)/t)^{\ell} \leq \epsilon_{\mathrm{p}}$. Indeed, for a PVW ciphertext $(\vec{a}, \vec{b})$ honestly encrypted under a key that is different (and independently and honestly generated) from pvwSK (generated uniformly randomly from $\mathbb{Z}_t^n$), $\vec{b} - \mathsf{pvwSK}^T\vec{a}$ is statistically close to a uniformly random distribution in $\mathbb{Z}_t^{\ell}$. The correctness of BFV introduces only negligible false positive rate.

Privacy: Analogously to Section 7.1, privacy follows from the key privacy of the PVW scheme used to generate clues (which is implied by LWE hardness, in turn implied by the assumed Ring-LWE hardness), and the semantic security of BFV encryption used by the detector (which is implied by Ring-LWE hardness). □

Asymptotic Complexity. When OMRp1 is instantiated with SRLC1, the asymptotic complexity is as follows. The digest size is $O(N + \log^2(\hat{k})\log(\epsilon_{\mathrm{n}}^{-1}))$ where the first term comes from the index retrieval and the second term comes from the number of combinations for payloads, but as $\hat{k} \leq N$, the cost is $O(N + \log^2(\hat{k})\log(\epsilon_{\mathrm{n}}^{-1}))$, which is not compact (though the constants are small). The computational complexity for the detector is $O(N(\log^2(\hat{k}) + \log(1/\epsilon_{\mathrm{p}})))$, as each message needs

[30]Note that r and ℓ are only used in Retrieve, so the detector can change these to improve efficiency if a recipient says that larger ϵ_{n} and ϵ_{p} are adequate. For simplicity, we do not include this flexibility in the pseudocode.

$\log(1/\epsilon_{\mathrm{p}})$ PVW ciphertexts as clues, and we have $\gamma = \log^2(\hat{k})\log(\epsilon_{\mathrm{n}}^{-1})$ when instantiated with $\mathsf{SRLC1}$ implied by Lemma 6.5. The recipient needs to decrypt the digest and thus have $O(N + \log^2(\hat{k})\log(\epsilon_{\mathrm{n}}^{-1}) + \hat{k}^3)$ computation complexity, to decrypt the $O(N + \log^2(\hat{k})\log(\epsilon_{\mathrm{n}}^{-1}))$ ciphertexts and then perform Gaussian elimination.

When $\mathsf{OMRp1}$ is instantiated with $\mathsf{SRLC2}$, the complexity is difficult to analysis. However, the computation and communication costs can be easily tested using empirical analysis under determined parameters, and our empirical analysis shows that $\mathsf{SRLC2}$ gives excellent concrete parameters (see Section 10.1).

Variable-Length Payloads. Another extension is allowing payloads of varying size (e.g., media messages). Our scheme is easily extended. The sender attaches a single clue to each message, regardless of length. The detector then divides each messages into segments of some fixed size (while padding the last segment, and adding a message identifier and a segment sequence number to enable reassembly), and then invokes the above scheme while reusing the message's clue for every segment of the message. The recipient, after decoding all segments of all pertinent messages, reassembles them. Privacy, soundness, and completeness are obviously maintained, except that $\bar{k}$ is now a bound on the total number of segments in pertinent messages, rather than number of pertinent messages.

Modulus Switching. Clue size can be reduced using LWE modulus switching with randomized rounding as shown in [DM15], if we set PVW secret keys to be ternary or binary (i.e., each secret key element is drawn from $\{-1,0,1\}$ or $\{0,1\}$). In $\mathsf{GenClue}$, every sender first uses the default q (which is typically large enough to guarantee r to be small, such that ϵ_{p} can be satisfied with a relatively small ℓ) to generate a PVW ciphertext, and then performs a modulus reduction from q to q' where $q' \ll q$, and then the original LWE noise is roughly scaled down by q'/q, and the effective new Gaussian noise has standard deviation of $O(\sqrt{||S||})$ where $||S||$ is the norm of the secret key. Then, the detector can use BFV to homomorphically compute the inner product between this modulus-reduced ciphertext and the secret key with a large BFV plaintext modulus (e.g., $t > 2(||S|| + 1)q'$), which then results in some number $uq' + e$ for pertinent messages, where $-||S|| < u < ||S||$ is an integer. The detector then performs a range check operation for every possible u homomorphically. Note that under carefully chosen $q, q', ||S||$, the new error can be much smaller than the error from public-key-based encryption directly using q', and therefore maintains a small ℓ to satisfy ϵ_{p} (recall that the detector running time is linear in ℓ). We do not include this optimization in our main constructions and implementation, since the use of low-weight secret keys interferes with DoS-resistance (see Section 8.2).

7.4 A Practical Compact OMR Scheme

While $\mathsf{OMRp1}$ above is practically efficient for many parameters of interest (cf. Section 10), its asymptotic digest size is still $O(N)$. An alternative approach achieves compactness, i.e., a digest size that grows only mildly with N when $\bar{k}$ is fixed and ϵ_{p} is small (cf. Definition 4.1). This can be achieved by using the *Randomized Digest Compression* approach of Section 7.2. The resulting practical and compact OMR algorithm, $\mathsf{OMRp2}$, is given in Algorithm 8

Theorem 7.2. *The scheme* $\mathsf{OMRp2}$ *in* $\mathsf{OMRp2}$ *in Algorithm 8 is a* OMR *scheme for* $N < D \cdot t/2$, *assuming security of* LWE *encryption (Section 5.3) and security of* BFV *leveled HE (Section 5.3.1), when instantiated with PRF* f *and an SRLC scheme* SRLC. *Moreover when instantiated with* $\mathsf{SRLC1}$, $\mathsf{OMRp2}$ *is also compact.*

Algorithm 8 $\mathsf{OMRp2}$: Practical Compact Oblivious Message Retrieval

```
1: procedure $\mathsf{OMRp2.GenParams}(1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n})$
2:     Choose $\mathsf{pp}_\mathsf{BFV} = (D, t, \dots)$, $\mathsf{pp}_\mathsf{PVW} = (n, w, \ell, q, \sigma)$, and range $r$ with one change:
3:     Replace item 3 with $\ell \cdot (1 - \mathsf{erf}(r/(\sqrt{2w}\sigma))) < \epsilon_\mathrm{n}/4$        ▷ See *Setting parameters*
4:     return $\mathsf{pp} = (1^\lambda, \epsilon_\mathrm{n}, \epsilon_\mathrm{p}, \mathsf{pp}_\mathsf{BFV}, \mathsf{pp}_\mathsf{PVW}, r)$        ▷ Provided implicitly below
5: procedure $\mathsf{OMRp2.KeyGen}$
6:     $(\mathsf{sk}_\mathsf{pvw}, \mathsf{pk}_\mathsf{pvw}) \leftarrow \mathsf{LWE.KeyGen}()$
7:     $(\mathsf{sk}_\mathsf{BFV}, \mathsf{pk}_\mathsf{BFV}) \leftarrow \mathsf{BFV.KeyGen}()$
8:     $\mathsf{ct}_\mathsf{pvwSK} \leftarrow \mathsf{BFV.Enc}(\mathsf{pk}_\mathsf{BFV}, \mathsf{sk}_\mathsf{pvw})$
9:     return $(\mathsf{sk} = (\mathsf{sk}_\mathsf{BFV}), \mathsf{pk} = (\mathsf{pk}_\mathsf{clue} = \mathsf{pk}_\mathsf{pvw}, \mathsf{pk}_\mathsf{detect} = (\mathsf{pk}_\mathsf{BFV}, \mathsf{ct}_\mathsf{pvwSK})))$
10: procedure $\mathsf{OMRp2.GenClue}(\mathsf{pk}_\mathsf{clue}, x)$
11:     $\vec{m} \leftarrow (0, \dots, 0) \in \mathbb{Z}_t^\ell$
12:     $c \leftarrow \mathsf{LWE.Enc}(\mathsf{pk}_\mathsf{clue}, \vec{m})$        ▷ Recall: clue $c \in \mathbb{Z}_t^{n \times \ell}$
13:     return $c$
14: procedure $\mathsf{OMRp2.Retrieve}(\mathsf{BB}, \mathsf{pk}_\mathsf{detect}, \bar{k})$
15:                                                        ▷ **Phase 1: Initialization**
16:     Draw a random seed $s = (s_f, s_h)$
17:     Parse $\mathsf{BB} = \{(x_1, c_1), \dots, (x_N, c_N)\}$ and $\mathsf{pk}_\mathsf{detect} = (\mathsf{pk}_\mathsf{BFV}, \mathsf{ct}_\mathsf{pvwSK})$
18:     Let $C \leftarrow N/(D \cdot \log(t))$
19:     Initialize $(\mathsf{Acc}_i = \mathsf{BFV.Enc}(\mathsf{pk}_\mathsf{BFV}, (0, \dots, 0)))_{i \in [C]}$        ▷ $D$ zeros
20:                                  ▷ **Phase 2: detection** in batches of $D$ messages
21:     $d \leftarrow \lceil 10\bar{k}/D \rceil \cdot D$ $C$ is smallest such that $1 - \prod_{i=1}^{\bar{k}-1}(1 - (\frac{i}{d})^C) < \epsilon_\mathrm{n}/4$
22:     $d'$ is the smallest integer such that $d'D \cdot \exp(-\frac{N(2d'-1)^2}{(2d'+1)d'D}) \leq \epsilon_\mathrm{n}/4$
23:     $d \leftarrow \mathsf{max}(d, d'D)$
24:     Initialize $(\mathsf{Acc}_{\mathsf{lhs},z} \leftarrow (\mathsf{BFV.Enc}(\mathsf{pk}_\mathsf{BFV}, (0, \dots, 0))_w)_{w \in [\log_t N]})_{z \in C}$
25:   ▷ $d$ zeros, encrypted into $\hat{d}$ accumulators, consisting of $\log_t N$ ciphertexts each with $D$ zeros
26:     Initialize $(\mathsf{Ctr}_{\mathsf{lhsCtr},z} \leftarrow \mathsf{BFV.Enc}(\mathsf{pk}_\mathsf{BFV}, (0, \dots, 0)))_{z \in C}$
27:     for $i = 1$ to $N/D$ do        ▷ Assume wlog that $D$ divides $N$
28:         Parse each clue as $c_{iD+i'} = ((c_{i',\kappa}))_{\kappa \in [n+\ell]} \in \mathbb{Z}_q^{n+\ell}$ for $i' \in [D]$
29:         Let $\bar{c}_\kappa = (c_{i',\kappa})_{i' \in [D]}$ for $\kappa \in [n+\ell]$ ▷ $\bar{c}_\kappa$ lists the $\kappa$-th element of every PVW clue in this
    batch
30:         $\alpha_1 \leftarrow \mathsf{InnerProd}(\mathsf{pp}_\mathsf{BFV}, \mathsf{pk}_\mathsf{BFV}, \mathsf{ct}_\mathsf{pvwSK}, (\bar{c}_\kappa)_{\kappa \in [n+\ell]})$
31:         $\alpha_2 \leftarrow \mathsf{RangeCheck}(\mathsf{pp}_\mathsf{BFV}, \mathsf{pk}_\mathsf{BFV}, \alpha_1, r)$
32:         $\alpha_3 = \prod_{i=0}^{\ell-1}(1 - \alpha_2[i])$
33:         $(\mathsf{PV}_{i'})_{i' \in ((i-1)D, iD]} \leftarrow \mathsf{PVUnpack}(\mathsf{pp}_\mathsf{BFV}, \mathsf{pk}_\mathsf{BFV}, \alpha_3)$
34:     for $i = 1$ to $N$ do
35:         $j, k \leftarrow f_{s_f}(i)$        ▷ $j \in [C]$, $k \in [\hat{d} \cdot D]$
36:         $\mathsf{Acc}_{\mathsf{lhs},j}[k] = \mathsf{Acc}_{\mathsf{lhs},j}[k] + i \cdot \mathsf{PV}_i$        ▷ t-ary addition
37:         $\mathsf{Ctr}_{\mathsf{lhs},j}[k] = \mathsf{Ctr}_{\mathsf{lhs},j}[k] + 1 \cdot \mathsf{PV}_i$        ▷ Slot-wise addition, done homomorphically
38:                                                        ▷ **Phase 3: Finalization**
39:     $\hat{k} \leftarrow \bar{k} + N\log(N)\epsilon_\mathrm{p}$
40:     $(\mathsf{pp}_\mathsf{SRLC}, m) \leftarrow \mathsf{SRLC.GenParams}(1^\lambda, \hat{k}, \epsilon_\mathrm{n}/4, t)$
41:         ▷ In practice $(\mathsf{pp}_\mathsf{SRLC}, m)$ is preprocesssed and tabulated and therefore becomes $O(1)$
42:     Initialize combinations $\{\mathsf{Cmb} = \mathsf{BFV.Enc}(\mathsf{pk}, 0)\}_{k \in [m]}$
43:     for $i = 1$ to $N$ do
44:         $S \leftarrow \mathsf{SRLC.GenWeights}(\mathsf{pp}_\mathsf{SRLC})$
45:         for $(j, w_j) \in S$ do
46:             $\mathsf{Cmb}_j = \mathsf{Cmb}_j + \mathsf{PV}_i \cdot x_i \cdot w_j$    ▷ by homomorphic addition and scalar multiplication
47:     return $M = (s, (\mathsf{Acc}_{\mathsf{lhs},i})_{i \in [C \cdot d/D]}, (\mathsf{Ctr}_{\mathsf{lhs},i})_{i \in [C \cdot d/D]}, (\mathsf{Cmb}_k)_{k \in [m]}, \mathsf{pp}_\mathsf{SRLC})$
48: procedure $\mathsf{OMRp2.Decode}(M, \mathsf{sk})$
49:     Similar to $\mathsf{OMRt1.Decode}$ in Algorithm 5, except that instead of a global counter $\mathsf{Ttl}$, it uses
    summation of individual counters.
```

Proof sketch . Completeness: Note that the false positive samples has negligible effect as in Theorem 6.2. Therefore, we assumes there are no false positives in our proof for simplicity.

A pertinent message is successfully retrieved if:

1. The clue's PVW ciphertext decrypts correctly using the range $[-r,r]$.
2. The detector's homomorphic evaluation, decrypted by the recipient, does not err. This is implied (up to negligible error) by the correctness of BFV.
3. Gaussian elimination succeeds (the equations are linearly independent)
4. No accumulator counter overflow (at most $t-1$ pertinent messages are mapped into each accumulator Acc_i)
5. The index is correctly retrieved through randomized index retrieval (assuming no overflow on counters.)

The first three conditions are similar as in Theorem 7.1, which adds up to error probability $2\epsilon_{\mathrm{n}}/4 + \mathsf{negl}(\lambda)$ (same as proof for Theorem 7.1). Condition 5 is satisfied with probability $\epsilon_{\mathrm{n}}/4$ by parameter choice in line 21.

For condition 4: note that we have $d = d'D$ buckets, each expected to be assigned at most N/d messages, as the number of messages that are detected as pertinent is trivially bounded by N. But bucket counters may overflow, i.e., get incremented by more than t assigned messages that are detected as pertinent (whether true positives or false positives). We bound this overflow probability as follows.

$$\begin{aligned}
\Pr[X \geq t] &< \Pr[X \geq 2N/D] \quad (\text{since } N < Dt/2)) \\
&= \Pr[X \geq 2(N/D)] \\
&= \Pr[X \geq 2(N/d)(d/D)] \\
&\leq \exp(-\frac{\delta^2}{2+\delta}\frac{N}{d}) \quad (\text{by Chernoff bound, where } \delta = 2(d/D) - 1 = 2d' - 1) \\
&\leq \exp(-\frac{(2d'-1)^2}{2d'+1}\frac{t/2}{d'})
\end{aligned}$$

By the union bound, the probability of none of the d buckets overflowing is $d\exp(-\frac{(2d'-1)^2}{2d'+1}\frac{t/2}{d'}) < \epsilon_{\mathrm{n}}/4$, where $d = O(\log(\epsilon_{\mathrm{n}}^{-1}))$. Therefore, for $N < Dt/2$, the condition at line 22 gives us a failure probability $< \epsilon_{\mathrm{n}}/4$.

Therefore, all five conditions together have a failure probability of $\epsilon_{\mathrm{n}} + \mathsf{negl}(\lambda)$ for $k \leq \bar{k}$

Soundness and Privacy: Follows exactly from Theorem 7.1.

Compactness: As we can see $D \cdot t = O(N)$, and therefore the digest size grows with $O(\mathsf{polylog} N)$. More specifically, it is $O(\hat{k}\log(\hat{k})\log(\epsilon_{\mathrm{n}}^{-1})\log^4(N))$, as we need $O(\log(N))$ accumulators for each Acc, and digest size grows with $\log(t)$ for each level of multiplication, $D = O(\log^2(t))$ because it grows with log(plaintext space) and number of multiplication levels (which also grows with $O(\log(t))$ due to $\mathsf{RangeCheck}$). Therefore, we obtain $\log^4(N)$ instead of $\log(N)$ as in Algorithm 5, but the rest are the same with the same analysis. And the updated $\hat{k} = \tilde{O}(\bar{k} + \epsilon_{\mathrm{p}}N)$. □

Computational Complexity. When OMRp2 is instantiated with SRLC1, its asymptotic complexity is as follows. The computational complexity for the detector is similarly computed as in Algorithm 5: $O(N(\log(\hat{k})\log(\epsilon_{\mathrm{n}}^{-1})\log^4(N)+\log(1/\epsilon_{\mathrm{p}})))$ when instantiated with SRLC1. The recipient's complexity is $O(\hat{k}\log(\hat{k})\log(\epsilon_{\mathrm{n}}^{-1})\log^4(N)+\hat{k}^3)$, where the first term is the size of the digest that needs to be decrypted, and the second term is the Gaussian elimination.

Similarly to OMRp1 above, if OMRp2 is instantiated with SRLC2, then the complexity is difficult to analysis, but can be tested empirically and yields excellent results.

In the expected parameter regime $\bar{k} \ll N < tD/2$ and $k \ll N$, OMRp2 can be more concretely efficient than the OMRp1, despite its additional complexity (cf. Section 10). Note, however, that as discussed in Section 7.2, for sufficiently large N (e.g., $N \gg tD$), the adversary may overflow counters by inducing an abnormally large number of pertinent messages (e.g., $k \geq tD$), in which case we need to use homomorphic big-integer counters, which worsens the concrete parameters.

7.5 Streaming Updates

If the board is large (e.g., millions of messages to be retrieved), then it would be preferable for the detector to process messages on-the-fly as they arrive, and be ready to serve the digest at low additional cost (and latency) when the recipient shows up . Moreover, we wish this processing to be doable even before knowing the number N of total messages and the bound $\bar{k}$ on number of pertinent messages, because the recipient may connect at unpredictable times and wish to catch up on all messages posted in the interim.

Our scheme has these properties, and to exploit them, the detector can break up OMR.Retrieve into several phases as annotated in Algorithm 7. After the initialization phase (lines 14–18), the messages can be processed in a streaming fashion for detection phase (lines 19–30), even though $\bar{k}$ (and the resulting m) are not yet known.

In the finalization phase (lines 31–39), the detector gets $\bar{k}$ and completes the computation by adding the payloads to their assigned combination ciphertexts. The finalization phase is much faster than the message processing phase, since it involves only relatively-inexpensive homomorphic operations (additions and scalar-by-ciphertext multiplications).

Seed Secrecy. If we assume that the board messages are not honestly generated, completeness of this scheme depends on the board messages being generated independently from the random seed s (otherwise, an adversary can cause decoding failures by crafting pertinent messages that predictably collide into the same slot(s), or whose weights predictably induce underdefined systems of equations). In a monolothic execution of OMR.Retrieve as in Algorithm 7, this is assured simply because s is randomly drawn *after* the board is given.

In streaming mode, we can instead assume that the detector keeps its seed s secret inbetween initialization and finalization (subsequent leakage is harmless). If this assumption is violated, then an adversary may indeed cause decoding failure; the recipient would detect this, and can then request a detection service from another detector discussed later in Section 7.7.

Alternatively, the detector can handle messages in batches of some size $\tilde{N}$, choosing a fresh seed for each batch after it is received. All $N/\tilde{N}$ seeds are then included in the digest, to allow the recipient to reproduce the slot assignments and weights. In practice, using 128-bit PRF seeds would insignificantly increase the digest size, if we naturally choose $\tilde{N} = D$ to fully utilize the BFV SIMD (where $D \geq 32768$ in our case).

7.6 Reducing Space Requirements

If the streaming update is never needed, we can slightly adjust Algorithm 7 (and similar to Algorithm 8), by performing $\mathsf{PVUnpack}$ on the fly to reduce memory. Instead of extracting all of $\mathsf{PV} = (\mathsf{PV}_1, \dots, \mathsf{PV}_N)$ at line 27, we can first process m_1 by computing PV_1, doing its index accumulation and linear combination of payloads, and then discarding m_1, PV_1 and the other intermediate values. We then proceed to similarly processes $m_2, \dots, m_N$. This greatly reduces the memory consumption, as each unpacked PV is a BFV ciphertext of size at least 60 kB.

If streaming updates are used, naively we need to store all the PV's together with its corresponding payload (or scan the board twice), since in Algorithm 7, phase 1 and 2 (lines 14–30) deal only with detection of clues, and phase 3 (lines 31–39) deals with retrieval of payloads. This PV requires large storage, which can be reduced by either of the following.

First, we can just save the packed PV's generated from line Algorithm 7. This is reduced storage, at the cost of slowing down phase 3 (since we need to finish the process after line Algorithm 7 and before phase 3).

Second, we can fix a generous $\bar{\bar{k}}$ which caps $\bar{k}$ (i.e., receivers can still choose their pertinent message bound $\bar{k}$ on the fly, but it can only be $\bar{k} \le \bar{\bar{k}}$). Then, the detector can proceed and finish $\mathsf{OMR.Retrieve}$ with $\bar{\bar{k}}$, and save the result. When getting $\bar{k}$ from the recipient, it can just "fold" the existing (too numerous) combinations prepared for $\bar{\bar{k}}$, by adding them into (fewer) combinations needed for $\bar{k}$ (see parameters at line 32). The folding is done by homomorphic additions of the BFV ciphertexts.

7.7 Handling Overflows

Retrieval may fail in case the number k of messages deemed pertinent overflows the bound $\bar{k}$: whether because of adversarial action, or because the recipient became unexpectedly popular. The possibility is inherent, since information-theoretically, a compact digest for a given $\bar{k}$ cannot represent $k \gg \bar{k}$ messages.[31] However, the recipient can robustly *detect* this case ($\mathsf{OMD.Decode}$ outputs $\mathsf{overflow}$) and act on it.

When an overflow is thus detected, the recipient can send another retrieval query to the detector, with a larger bound $\bar{k}$. However, this may create an information leak: the detector, observing the repeated query, could deduce that this particular recipient is popular (has many pertinent messages) among the processed set of messages, and thus deduce information about the recipient's identity or message traffic. To prevent the linkability of the two queries, the recipient could try to issue the second query from a fresh network connection using a different IP address (e.g., using a new Tor connection), but the repeated detection key would still be noticed. Likewise, the recipient may attempt to use two independent detection servers, but the two may still collude (or be observed) and notice the reoccuring detection key. As discussed in Section 9, it is possible to avoid this leakage thorough the *full-key-unlinkability* property of these constructions.

Counting Queries. Instead of repeating retrieval queries with enlarged $\bar{k}$, the recipient can register to two servers: the first to do lightweight detection (without retrieval) just to obtain the count of pertinent messages k, and the second to do full retrieval using $\bar{k} = k$ (or some noisy version thereof to hide the exact k). The first query can be further compacted by having the detector send just the sum of the PV_i's (computed in batches of size t to avoid counter overflows).

[31] Overflow could also occur because of an excessive number of false positives in PV, but this is capped by the soundness analysis, and we increase $\bar{k}$ to $\hat{k}$ to compensate for it.

7.8 Detection Key Size Reduction

The detection key includes the BFV ciphertexts $\mathsf{ct}_{\mathsf{pvwSK}}$ encrypting $\mathsf{sk}_{\mathsf{pvw}}$, and the BFV public keys (including encryption key, relinearization key, and rotation keys as in [Lai, §5.6] for rotation operations as in [SV14, BGV12, LPR13]). Their size is $O(1)$, but concretely quite large (cf. Section 10.2).[32] We can reduce it in several ways.

First, all of the aforementioned components are RLWE ciphertexts of the form $(\vec{a}, \vec{b})$, generated by the recipient who knows the corresponding RLWE secret key, and who can thus choose a pseudorandom $\vec{a}$ that is represented as a short PRG seed, thereby halving the ciphertext size.[33]

Second, we pack the $n \cdot \ell$ elements of $\mathsf{ct}_{\mathsf{pvwSK}}$ into ℓ BFV ciphertexts, and modify the $\mathsf{InnerProd}$ accordingly, as follows. For the simplest case of $\ell = 1$ and PVW secret key $\mathsf{sk} \in \mathbb{Z}_q^n$ where n divides D (or padded to such): in the BFV ciphertext ct encrypting sk, the i-th slot encrypts $\mathsf{sk}_{i \bmod n}$ for $i \in [D]$. Homomorphically compute ct^i, which is ct with its slots rotated to the left i times, for $i \in [0, n-1]$. For D clues at a time, denote the i-th clue by $(\vec{a}_i, \vec{b}_i)$. Homomorphically compute $\mathsf{ct}^{i\prime} \leftarrow \mathsf{ct}^i \cdot (\vec{a}_j[j + i \bmod n])_{j\in[n]}$ and $\mathsf{ct}' \leftarrow \sum_{i=0}^{n-1} \mathsf{ct}^{i\prime}$. Now, ct' encrypts the result of $(\mathsf{PVW.sk})^T \vec{a}_i$ in its i-th slot. This trivially extends to $\ell > 1$ by repeating this process ℓ times. Note that the rotation, multiplication, and summation can be done sequentially (i.e., rotate by 1, multiply, and then sum to ct' for one iteration at a time), so we only need to have one ciphertext in memory at a time.

Third, the detection key size is dominated by the row rotation keys used for the homomorphic evaluation of slot rotations. In Algorithm 7 and Algorithm 8, we use keys for all rotation-by-power-of-2, but since this happens near the end of the homomorphic evaluation, we minimize their size by creating rotation keys that support only 3 more multiplicative levels. A top-level rotation key is included just for rotation by 1, as needed by previous optimization.

The detection key also needs to be *stored* by the detector, so the above also reduces the storage/memory requirements (especially if decompression is done on-the-fly and amortized across many messages and cores).

8 Denial-of-Service Resistance

Thus far we have assumed, as in prior works, that all clues in the board are generated honestly by the prescribed $\mathsf{GenClue}$ algorithm, using various honestly-generated clue keys. However, this may be violated in reality. Clues may be generated incorrectly, or even maliciously, especially if anyone is allowed to add messages to the board (as in blockchain applications).

In a *Denial of Service (DoS) attack* on an OMR or OMD scheme, the adversary can maliciously generate any of the clues $c_i \in \mathcal{C}$ in board messages, in an attempt to induce false positives or false negatives in the subsequent detection/retrieval. The adversary could simply use this power to create pertinent messages for some recipient, thus trying to induce an overflow for that recipient (by exceeding their $\bar{k}$); this is inevitable and handled in Section 7.7. But the bigger danger is *amplified DoS*: it could also be the case that even a single maliciously-crafted clue, placed in the board, will cause catastrophic failure (e.g., causing *many* recipients to overflow or miss pertinent messages)

[32]The digest size is independent of N and $\bar{k}$. Naively, it grows as log-logarithmically with ϵ_{p} and ϵ_{n} due to the multiplicative depth of the evaluated circuit, but asymptotically we can cap this growth using BFV bootstrapping [CH18, KDE+21].

[33]This is SEAL's seeded secret-key encryption mode, see `https://github.com/microsoft/SEAL/blob/main/native/examples/6_serialization.cpp`.

or soundness (e.g., causing *many* recipients to misdetect messages as pertinent). Furthermore, the adversary may also take the role of a recipient in the system, and publish a maliciously-crafted clue key, thereby inducing honest senders to unwittingly generate harmful clues.

As shown next, the above attacks are possible in natural and prior schemes, but our OMRp1 and OMRp2 schemes are resilient to these.

8.1 Threat Model (DoS)

In the DoS threat model, the computationally bounded adversary has all the power as defined in Section 4.2. Additionally, it is allowed to generate any (perhaps malformed) clues and post them on the board, as well as generate any (perhaps malformed) clue keys for other senders to use. Thus, for correctness and soundness, we assume only that the detector is honest but curious. Other parties are malicious. As before, for privacy, everyone but the message's sender and recipient are assumed malicious and colluding.

8.2 Simple Attacks

Attack on the Simple FHE Scheme. Consider the generic FHE-based scheme of Algorithm 5, instantiated with FHEW or TFHE (the technique may be extended differently depending on schemes). Then, a *malicious sender* can generate a *wildcard ciphertext* that decrypts to 1 for almost any honesty-generated key pair. By setting all the ℓ clue ciphertexts to wildcard ciphertexts, the adversary will cause the message including this clue to be detected as pertinent for almost *all* recipients, and if there are many such messages, then these recipients would all overflow.

Specifically, recall that FHEW and TFHE use a low-weight secret key sk; for simplicity assume it is binary: $\mathsf{sk} \in \{0,1\}^n \subset \mathbb{Z}_q^n$ (the argument generalizes to ternary etc.). Then one wildcard ciphertext is $\mathsf{ct} = (\vec{a}, b)$, where $\vec{a} = 0^n$ and $b = 0$, which always decrypts to 0 under any FHEW/TFHE public key. Another wildcard ciphertext is $\mathsf{ct} = (\vec{a}, b)$, where $\vec{a} = (1, \ldots, 1) \in \mathbb{Z}_q^n$ and $b = n/2 + q/2 \in \mathbb{Z}_q$, which is very likely to decrypt to 0.[34] This generalizes.

Worse yet, even if all the *senders* are generating their clues honestly, a DoS attack can be initiated by a *malicious receiver*. Such a receiver can publish a clue key that is all-zero, and induce any sender to send some message to that clue key. The resulting clue will be all zero (since the sender's GenClue honestly combines clue-key elements), which is one of the aforementioned wildcard ciphertexts.

Fuzzy Message Detection. The FMD schemes of [BLMG21] are similarly vulnerable, as also observed in [Lew21b, commit `e19b99112e`] for some choices of clues (all zeros or all ones). In addition, for some other choices of underlying asymmetric key encryption, for example, FHEW and TFHE (which fulfill the CPA security requirements of these FMD schemes), the above wildcard ciphertext attack carries over. These attacks may be generalized and therefore the mitigation is non-trivial. These clue would cause messages to be misperceived as pertinent by almost *all* recipients, thereby increasing their communication and computation cost.

[34]The Hamming weight of sk, and thus $\langle \vec{a}, \mathsf{sk} \rangle$, has a mean of $n/2$ and standard deviation $\sqrt{n/12}$. Thus, it holds that $\langle \vec{a}, \mathsf{sk} \rangle - b \in q/2 \pm q/4$ with probability $\approx \mathsf{erf}(\frac{q/4}{\sqrt{2n/12}})$ over sk, in which case the decryption result is 1. For example, with the common parameters $n = q = 512$ [PAL21], the probability of decrypting to 1 is $\approx 1 - 10^{-53}$, so even for $\ell = 20$ ciphertexts per clue, all will decrypt to 1 with probability $> 1 - 10^{-51}$.

Private Signaling. The Private Signaling scheme of [MSS^{+}21] is also vulnerable to DoS attacks. The single-server scheme is vulnerable to wildcard ciphertext attacks, which can overflow recipients as above. Moreover, their model, where signals are sent to servers rather than placed on the board, opens an additional DoS attack vector. For instance, a malicious sender can send many pertinent-looking signals to a server (or, in the two-servers scheme, to a server pair) without putting anything on the board, and therefore cause extra work for the server(s) to overflow the message accumulator for recipients.[35]

8.3 DoS Resistance Definition

We introduce a formal definition of DoS resistance, strengthening the OMR definition of Section 4.3. Recall that there, Definition 4.1provides soundness and completeness guarantees only when the clues in the board are honestly generated, using clue keys which are themselves honestly generated. In that case, there is a natural ground-truth notion of pertinent messages, defined by which clue key $\mathsf{pk}_{\mathsf{clue}}$ each clue was generated for; hence soundness and completeness are defined in reference to that ground truth.

Now, however, we wish to capture a stronger notion, where clues may be maliciously generated and may not obviously correspond to any specific clue key (e.g., consider the wildcard ciphertext above). We thus require the existence of an *indicator* predicate $\mathcal{I}(c,\mathsf{pk}_{\mathsf{clue}})$ that serves as a ground truth for whether a given clue c is pertinent to a given user specified by their clue key $\mathsf{pk}_{\mathsf{clue}}$. This predicate, which may not be efficiently computable, should give the natural answer for honestly-generated clues. For otherwise-generated clues, the indicator may answer arbitrarily, except that it must still make up its mind, i.e., not claim more than one honest recipient as the intended one, except with small probability. This *collision resistance* property means that, while a malicious sender can (inevitably) craft a message that is be considered pertinent by one user, it is difficult to spam *multiple* users with a single message.

Soundness and completeness are then redefined w.r.t the indicator $\mathcal{I}$, as below. Note that to facilitate tight analysis, the completeness (false negative rate) bound ϵ_{n} in the definition is broken up into two components: the rate ϵ_{i} at which the indicator fails to detect truly pertinent messages (which may be non-negligible because a indicator with high thresholds may help achieve collision resistance), and the rate $\epsilon_{\mathrm{n}} - \epsilon_{\mathrm{i}}$ at which the scheme fails to retrieve messages flagged by the indicator (which may be on-negligible because of error sources in the concrete scheme).

Definition 8.1 (DoS-resistant OMR)**.** Let OMR be an OMR scheme for error rates $\epsilon_{\mathrm{n}}, \epsilon_{\mathrm{p}}$ (as in Definition 4.1). An *indicator* with an *indicator false negative rate* $\epsilon_{\mathrm{i}} \leq \epsilon_{\mathrm{n}}$ for OMR is a function $b \leftarrow \mathcal{I}(\mathsf{pp}, x, c, \mathsf{pk}_{\mathsf{clue}}, \mathsf{sk})$ on a public parameter pp, a message (x, c), a clue key $\mathsf{pk}_{\mathsf{clue}}$, and its corresponding secret key sk, outputs $b \in \{0,1\}$, such that:

[35]In the SGX-based single-server scheme, this could be prevented by placing signals publicly within the board. In the GC-based two-server scheme, the signal shares must not be publicly revealed, but the commitments thereto can be put on the board. This requires adding information ("clues") to the board, as in our OMD/OMR model. Alternatively, the servers can check the board, and ignore signals that do not correspond to new board messages. But this works well only if a single SGX-based server (or a GC-based server pair) handles all users; otherwise, an attacker can put one new message on the board, and then send signals to *every* server (or server pair) purporting to pertain to that message, while equivocating by sending different signals, so that *every* server detects that message as pertinent for one of its own users. Moreover, this mitigation does not apply to the two-server (GC) based solution, as the signals sent to each server needs to be kept private.

- *(Indicator completeness)* For $\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n})$, honest-generated key pair $(\mathsf{sk}, \mathsf{pk} = (\mathsf{pk}_\mathsf{clue}, \cdot)) \leftarrow \mathsf{KeyGen}()$, for any payload x, and honest-generated clue $c \leftarrow \mathsf{OMR.GenClue}(\mathsf{pk}_\mathsf{clue}, x)$, it holds that
$$\Pr[\mathcal{I}(\mathsf{pp}, x, c, \mathsf{pk}_\mathsf{clue}, \mathsf{sk}) = 1] \geq 1 - \epsilon_\mathrm{i} - \mathsf{negl}(\lambda) \ .$$
- *(Collision resistance)* For any PPT adversary $\mathcal{A}$, let $\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n})$, two honest-generated key pairs $(\mathsf{sk}, \mathsf{pk} = (\mathsf{pk}_\mathsf{clue}, \cdot)) \leftarrow \mathsf{OMR.KeyGen}()$ and $(\mathsf{sk}', \mathsf{pk}' = (\mathsf{pk}'_\mathsf{clue}, \cdot)) \leftarrow \mathsf{OMR.KeyGen}()$, and adversarially-generated $(x, c) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{pk}')$, for $b \leftarrow \mathcal{I}(\mathsf{pp}, x, c, \mathsf{pk}_\mathsf{clue}, \mathsf{sk})$ and $b' \leftarrow \mathcal{I}(\mathsf{pp}, x, c, \mathsf{pk}'_\mathsf{clue}, \mathsf{sk}')$, it holds that
$$\Pr[b = 1 \ \wedge \ b' = 1] \leq \epsilon_\mathrm{p} + \mathsf{negl}(\lambda) \ .$$

An OMR scheme OMR is *DoS-resistant* for ϵ_n and ϵ_p if there exists an indicator $\mathcal{I}$ with an indicator false negative rate ϵ_i for OMR such that for any PPT adversary $\mathcal{A}$, for $\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n})$, $(\mathsf{sk}, \mathsf{pk} = (\mathsf{pk}_\mathsf{clue}, \mathsf{pk}_\mathsf{detect})) \leftarrow \mathsf{OMR.KeyGen}()$, and adversarially-generated board $\mathsf{BB} \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pk})$ where $\mathsf{BB} = (\,(x_1, c_1), \ldots, (x_N, c_N)\,)$ and $(x_i)_{i \in [N]}$ are unique, for any $0 < \bar{k} \leq N$, letting $M \leftarrow \mathsf{Retrieve}(D, \mathsf{pk}_\mathsf{detect}, \bar{k})$, $\mathsf{PL} \leftarrow \mathsf{Decode}(M, \mathsf{sk})$:

- *(DoS-completeness)* Let $k = \sum_{i=0}^{N} \mathcal{I}(\mathsf{pp}, x_i, c_i, \mathsf{pk}_\mathsf{clue}, \mathsf{sk})$. Then either $k > \bar{k}$ and $\mathsf{PL} = \mathsf{overflow}$, or $\Pr[x_j \in \mathsf{PL} \mid \mathcal{I}(\mathsf{pp}, x, c, \mathsf{pk}_\mathsf{clue}, \mathsf{sk}) = 1] \geq 1 - (\epsilon_\mathrm{n} - \epsilon_\mathrm{i}) - \mathsf{negl}(\lambda)$ for all $j \in [N]$.
- *(DoS-soundness)* $\Pr[x_j \in \mathsf{PL} \mid \mathcal{I}(\mathsf{pp}, x, c, \mathsf{pk}_\mathsf{clue}, \mathsf{sk}) = 0] \leq \mathsf{negl}(\lambda)$ for all $j \in [N]$.

Note that DoS-completeness implies the (weaker) completeness of Definition 4.1 with the same false negative rate ϵ_n, and DoS-soundness implies the (weaker) soundness of Definition 4.1 with false positive rate $\epsilon_\mathrm{p} + \epsilon_\mathrm{n}$. Completeness is trivial, and soundness is given by the following lemma:

Lemma 8.2. *Any tuple of algorithms* OMR *that is* ϵ_p*-DoS-sound by Definition 8.1 is* $(\epsilon_\mathrm{p}+\epsilon_\mathrm{n})$*-sound by Definition 4.1.*

Proof. Let $\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n})$. Let a board BB, a set S of pertinent messages, and a key pair $(\mathsf{sk}, \mathsf{pk} = (\mathsf{pk}_\mathsf{clue}, \mathsf{pk}_\mathsf{detect}))$ be generated as in Definition 4.2 for any choice of p, partition and payloads therein.

If $j \notin S$, then, for (x_j, c_j) there exist a different public key pk' such that c was honestly generated as $c \leftarrow \mathsf{GenClue}(\mathsf{pk}'_\mathsf{clue}, x)$ by Definition 4.2. By indicator completeness, $\Pr[\mathcal{I}(\mathsf{pp}, x, c, \mathsf{pk}'_\mathsf{clue}, \mathsf{sk}') = 1] \geq 1 - \epsilon_\mathrm{n} - \mathsf{negl}(\lambda)$, an therefore by collision resistance, $\Pr[\mathcal{I}(\mathsf{pp}, x, c, \mathsf{pk}_\mathsf{clue}, \mathsf{sk}) = 1] \leq \epsilon_\mathrm{p}(1 - \epsilon_\mathrm{i}) + \epsilon_\mathrm{i} + \mathsf{negl}(\lambda) \leq \epsilon_\mathrm{p} + (1 - \epsilon_\mathrm{p})\epsilon_\mathrm{i} + \mathsf{negl}(\lambda) < \epsilon_\mathrm{n} + \epsilon_\mathrm{p} + \mathsf{negl}(\lambda)$. Therefore, $\Pr[x_j \in \mathsf{PL}] \leq \epsilon_\mathrm{n} + \epsilon_\mathrm{p} + \mathsf{negl}(\lambda)$, which satisfies the soundness definition in Definition 4.1. □

DoS-resistant Oblivious Message Detection. The OMD definition can be strengthened analogously.

Generic ZK Solution? A natural attempt to upgrade any OMR scheme to become DoS-resistant is to accompany each clue with a zk-SNARK proof [Mic00, BCCT12] that the clue was correctly generated (from a clue key that was also correctly generated, using recursively composed ZKPs [Val08, BCCT13]). However, this is not in general adequate, because a malicious sender (or clue key generator) could skew the randomness fed into $\mathsf{GenClue}$ (or KeyGen) to induce soundness or completeness failures that would have low probability in the honest case.

8.4 Attaining DoS-resistant OMR

The $\mathsf{OMRp1}$ scheme of Section 7 already satisfies DoS resistance (for $\epsilon_\mathrm{p} = \mathsf{poly}(\lambda)$, with a minor change, under the natural computational assumption stated below). Intuitively, this is because PVW encryption has the property that a ciphertext that decrypts to 0 can be generated by adding up columns of the public key, but if the ciphertext is efficiently generated in any *other* way (e.g., from a different public key, or "out of the blue"), then its decryption is close to uniformly random.

Patched $\mathsf{OMRp1}$. The exception to the above intuition is trivial ciphertexts (i.e., adding up *none* of the public-key columns), so we redefine the clue space as $\{(\vec{a},\vec{b}) \in \mathbb{Z}_q^{n+\ell} : \vec{a} \neq 0^n\}$. Accordingly, we change $\mathsf{OMRp1.Retrieve}$ (at line 15) to reject clues where $\vec{a} = 0^n$. Moreover, we change $\mathsf{OMRp1.GenClue}$ such that if generates a clue with $\vec{a} = 0^n$, it retries with fresh randomness (and aborts after λ attempts).

Theorem 8.3. *For any $\epsilon_\mathrm{p} = \mathsf{poly}(\lambda)$, Algorithm 7 (patched as above) is a DoS-resistant Oblivious Message Retrieval scheme, when instantiated with any PRF f, assuming the hardness of Ring-LWE and Conjecture 8.4 below.*

The requisite Conjecture 8.4 , stated below, is a new but natural conjecture about the behavior of Regev05 encryption (i.e., LWE samples). Crucially, it is needed only to prove the DoS-resistance of soundness and completeness. Privacy, as well as non-DoS soundness and completeness, rely only on the standard Ring-LWE assumption (cf. Theorem 7.1).

Proof sketch . We now prove that our construction satisfies DoS Resistance of OMR according to our definition 8.1.

First, an indicator is constructed as follows. After taking clue $c = (\vec{u}_a, \vec{u}_b) \in \mathcal{C}$, it computes $\vec{u} = \vec{u}_b - \mathsf{sk}^T\vec{u}_a$. Then, it returns 1 iff for all $u_i \in \vec{u}$, $u_i \in [-r, r]$, and outputs 0 otherwise. This is complete because if a clue $c \leftarrow \mathsf{GenClue}(\mathsf{pk}_{\mathsf{clue}}, x)$, clue c encrypts 0^ℓ and each of the ℓ elements has a Gaussian noise with standard deviation of $\sqrt{w}\sigma$. Then, as $\ell \cdot (1 - \mathsf{erf}(r/(\sqrt{2w}\sigma))) \leq \epsilon_\mathrm{n}/2$, the indicator completeness is satisfied with $\epsilon_\mathrm{i} = \epsilon_\mathrm{n}/2$. Indicator collision resistance follows from Conjecture 8.4 and Lemma 8.5 below.

Note that $\mathsf{OMRp1.Retrieve}$ performs the same computation as the above indicator, but under BFV homomorphic evaluation. Thus, DoS-soundness is trivial: for any clue c_i, $i \in [N]$, such that $\mathcal{I}(\mathsf{pp}, x_i, c_i, \mathsf{pk}_{\mathsf{clue}}, \mathsf{sk}) = 0$, we have $\mathsf{PV}_i \in \langle 0 \rangle^{\mathsf{BFV}}_{\mathsf{sk}}$ (with only negligible probability of BFV failure), and thus the message will not be added to PL.

Similarly, for DoS-completeness: for any clue c_i, $i \in [N]$, such that $\mathcal{I}(\mathsf{pp}, x_i, c_i, \mathsf{pk}_{\mathsf{clue}}, \mathsf{sk}) = 1$, we have $\mathsf{PV}_i \in \langle 1 \rangle^{\mathsf{BFV}}_{\mathsf{sk}}$ (with negligible error probability due to BFV failure). If $k < \bar{k}$, the only other source of getting $\mathsf{PL} = \mathsf{overflow}$ is from our SRLC method, and as shown at line 32, it holds enough combinations to make the retrieval failure bounded by $\epsilon_\mathrm{n}/2 + \mathsf{negl}(\lambda) = \epsilon_\mathrm{n} - \epsilon_\mathrm{i} + \mathsf{negl}(\lambda)$. (Note that the completeness of SRLC is unaffected by adversarial input, so its average-case guarantee suffices.) For $k > \bar{k}$, our deterministic counter gives k and outputs $\mathsf{overflow}$ accordingly.

Lastly, $\mathsf{OMRp1.GenClue}$ is still PPT despite resampling with fresh randomness when the generated clue has $\vec{a} = 0^n$. For any honestly generated clue key pk, by [PVW08, Lemma 7.4], $(\vec{a},\vec{b}) \leftarrow \mathsf{PVW.Enc}(\mathsf{pk}, \vec{0})$ has $\vec{a}$ in a distribution statistically indistinguishable from the uniform distribution in $\mathbb{Z}_q^n$, thus $\vec{a} = \vec{0}$ with probability $\leq q^{-n} + \mathsf{negl}(\lambda)$. The probability that all λ trails has clue output as $\vec{a} = 0^n$ is thus $n^{-\lambda \cdot q} + \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda)$. $\square$

Snake-Eye Conjecture. To argue that the above indicator is collision resistant, we will rely on the following natural conjecture about LWE-based encryption. Phrased in terms of the Regev05 [Reg09] encryption scheme (which is identical to PVW for $\ell = 1$), it says: it is infeasible, given two honestly-generated Regev05 public keys, to find a nontrivial ciphertext that decrypts to 0 under both of the corresponding secret keys (we call such a ciphertext a *snake-eye*), except with a negligible advantage over trivial (i.e., $1/2 + \mathsf{negl}$, attained by just encrypting 0 to the first key). Moreover, when Regev05 decryption is changed to use the range $\pm r$ instead of $\pm q/4$, for any $1 \leq r < q/4$, then no adversary can find a snake-eye with probability better than the trivial $(2r+1)/q + \mathsf{negl}$. Formally stated:

Conjecture 8.4 (Regev05 is snake-eye resistant)**.** For any PPT algorithm $\mathcal{A}$, for Regev05 encryption with modulus q and remaining parameters for which semantic security holds, for any $1 \leq r < q/4$, for key pairs $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $(\mathsf{sk}', \mathsf{pk}') \leftarrow \mathsf{KeyGen}(1^\lambda)$, for ciphertext $(\vec{a}, b) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{pk}', r)$, it holds that

$$\Pr\left[\begin{array}{c} |\langle \vec{a}, \mathsf{sk}\ \rangle + b| \leq r \\ \wedge\ |\langle \vec{a}, \mathsf{sk}' \rangle + b| \leq r \end{array} \ \wedge\ \vec{a} \neq \vec{0}\right] \ \leq\ (2r+1)/q + \mathsf{negl}(\lambda)\ .$$

Snake-eye resistance is not generically implied by semantic security of the encryption scheme, nor by key privacy.[36] However, Conjecture 8.4 can be proven under the standard (homogeneous) *Short Integer Solution (SIS)* hardness assumption [Ajt96] combined with a natural generalization of the *Knowledge of Knapsack of Noisy Inner Products* assumption [BCCT12]. Alternatively the latter assumption can be replaced by a zk-SNARK proof [Mic00, BCCT12], appended to the ciphertext, of the statement "$(\vec{a}$,$b)$ was constructed as a linear combination of public-key vectors".[37]

Snake-eye resistance naturally generalizes to PVW encryption, saying that for PVW with plaintext space $\mathbb{Z}_q^\ell$, the probability of producing a snake-eye (a ciphertext that decrypts to all-zero-vectors under two given public keys) shrinks exponentially in ℓ. For small ℓ (which suffices for us), this follows from Conjecture 8.4 by a hybrid argument with sampling rejection:

Lemma 8.5 (PVW is snake-eye resistant)**.** *Under Conjecture 8.4 , for any PPT adversary $\mathcal{A}$, for* PVW *encryption with modulus q and plaintext space $\mathbb{Z}_2^\ell$, and r such that $((2r+1)/q)^{-\ell} = \mathsf{poly}(\lambda)$ and remaining parameters for which semantic security hold, for any $1 \leq r < q/4$, for key pairs* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{PVW.KeyGen}()$ *and* $(\mathsf{sk}', \mathsf{pk}') \leftarrow \mathsf{PVW.KeyGen}()$*, for ciphertext $c = (\vec{a}, \vec{b}) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{pk}', r)$, letting $\vec{m} \leftarrow \mathsf{sk}^T A + \vec{b}$ and $\vec{m}' \leftarrow \mathsf{sk}'^T \vec{a} + \vec{b}$, it holds that:*

$$\Pr\Big[(\ \forall i \in [\ell] : |m_i| \leq r \wedge |m_i'| \leq r\ ) \wedge\ \vec{a} \neq \vec{0}\Big] \ \leq\ ((2r+1)/q)^\ell + \mathsf{negl}(\lambda)\ . \qquad (2)$$

Proof sketch . By induction on ℓ. The base case $\ell = 1$ is identical to Conjecture 8.4 .

Suppose the lemma holds for some ℓ but not for $\ell + 1$. Then there exists a PPT algorithm $\mathcal{A}$ such that for ciphertext $c = (\vec{a}, \vec{b}) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{pk}', r)$, letting $\vec{m} \leftarrow \mathsf{sk}^T A + \vec{b}$ and $\vec{m}' \leftarrow \mathsf{sk}'^T \vec{a} + \vec{b}$, it

[36]Any such secure encryption can be "spoiled" by adding a special ciphertext that always decrypts to zero but is never generated by Enc, thereby maintaining security but breaking snake-eye resistance.

[37]Using the Groth16 scheme [Gro16], the size of this proof is 192 bytes [HBHW21, §5.4.9.2] per clue regardless of ℓ, and in Zcash the statement can be merged into pre-existing zk-SNARK proofs in the same transaction [HBHW21].

holds that:

$$\begin{aligned}
&((2r+1)/q)^{\ell+1} + \mathsf{nonnegl}(\lambda) \\
<& \Pr\left[(\,\forall i \in [\ell+1] : |m_i| \le r \wedge |m'_i| \le r\,) \wedge\ \vec{a} \ne \vec{0}\right] && \text{(Eq. (2) does not hold for } \ell+1\text{)} \\
=& \Pr\Big[\underbrace{\Big((\,\forall i \in [\ell] : |m_i| \le r \wedge |m'_i| \le r\,) \wedge\ \vec{a} \ne \vec{0}\Big)}_{A_\ell} \wedge \underbrace{\Big(|m_{\ell+1}| \le r \wedge |m'_{\ell+1}| \le r\Big)}_{B_{\ell+1}}\Big] \\
=& \Pr\left[A_\ell \wedge B_{\ell+1}\right] \ = \ \Pr[A_\ell] \cdot \Pr[B_{\ell+1}|A_\ell] \\
\le& \big(((2r+1)/q)^{\ell} + \mathsf{negl}(\lambda)\big) \cdot \Pr[B_{\ell+1}|A_\ell] && \text{(Eq. (2) holds for } \ell\text{)}
\end{aligned}$$

It follows that $\Pr[B_{\ell+1}|A_\ell] > (2r+1)/q + \mathsf{nonnegl}(\lambda)$. We will use this to construct a PPT algorithm $\mathcal{A}'$ that violates Conjecture 8.4 .

The algorithm $\mathcal{A}'$, given a challenge ($\mathsf{pk}^* \in \mathbb{Z}^{(n+1)\times w}, \mathsf{pk}'^* \in \mathbb{Z}^{(n+1)\times w}$), creates a snake-eye ciphertext as follows. Let $\mathsf{pk}_n \in \mathbb{Z}^{n\times w}$ denote the first n rows of pk^* (i.e., the vector parts of each of the w LWE sample in pk^*). Generate ℓ new PVW secret keys $\mathsf{sk} = (\mathsf{sk}_i)_{i\in[\ell]}$. Let pk denote pk^* extended with ℓ new rows, containing $\mathsf{sk}^T\mathsf{pk}_n$, inserted before the last row (so that pk looks like a PVW public key for plaintext space $\mathbb{Z}_2^{\ell+1}$). Let sk', pk'_n and pk' be defined analogously from pk'^*. Then, run $c = (\vec{a}, \vec{b}) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{pk}')$. Compute $\vec{m} = \mathsf{sk}^T A + \vec{b}$ and $\vec{m}' = \mathsf{sk}'^T\vec{a} + \vec{b}$. If condition A_ℓ (defined above) holds, output $c = (\vec{a}, b_{\ell+1})$; otherwise repeat with freshly drawn $\mathsf{sk}, \mathsf{sk}'$.

Note that the distribution of (pk,pk',sk,sk') induced by $\mathcal{A}'$ is identical to what $\mathcal{A}$ expects above, and thus here too, $\Pr[A_\ell] > ((2r+1)/q)^{\ell+1}$, so the expected number of iterations is $\mathsf{poly}(\lambda)$ as $((2r+1)/q)^{-\ell} = \mathsf{poly}(\lambda)$. Similarly, here too $\Pr[B_{\ell+1}|A_\ell] > (2r+1)/q + \mathsf{nonnegl}(\lambda)$, and thus once $\mathcal{A}'$ terminates, event $B_{\ell+1}$ also holds with that probability, in which case c is a snake-eye ciphertext for the challenge (pk,pk^*), contradicting Conjecture 8.4 . □

This immediately implies the indicator collision resistance property above.

Analogously, OMRp2 in Algorithm 8 is likewise a DoS-resistant OMR after patching the clue space.

9 Key Unlinkability

9.1 Key Linkability Issues

The security notions discussed thus far, and in prior works, omit another privacy consideration: linkability of the detection and clue keys. This occurs in several senses:

Detection-Key to Clue-Key Linkability. Recall that the clue key may be publicly associated with the recipient's identity (e.g., a private messaging address may be shared; or a cryptocurrency payment address, augmented with the clue key as in Section 11, may be publicly posted). Thus, if the detection key is the same as the clue key, or otherwise linkable to it, then the recipient is effectively disclosing their identity to the detector, and thereby perhaps to the whole world (since in our threat models, detectors may be unwilling or unable to keep secrets). Consequentially, recipients may be inadvertently broadcasting the time when they go online to check for new messages, along with their choice of pertinent-message bound $\bar{k}$, and (in the absence of adequate network-level anonymity) IP address — all tied to their identity.

Detection-Key to Detection-Key Linkability. Another privacy concern is that multiple detection queries may be linkable to each other, thus allowing traffic analysis of detection queries.

Clue-Key to Clue-Key Linkability. Recipients may wish to publish several addresses that are unlinkable, but (secretly) correspond to a single secret key which controls all of them (e.g., "diversified address" in Zcash [HBHW21]). This seems incompatible with the simple detection model discussed so far: if all of these addresses carry the same clue key, then they become trivially linkable by that; whereas if they carry fresh clue keys, then the messages pertinent to each have to be retrieved separately, with corresponding growth in cost.

Linkability in Prior Work. In FMD [BLMG21], the detection keys and clue keys are a key pair in an asymmetric encryption scheme (i.e., $\mathsf{pk}_{\mathsf{detect}} = \mathsf{sk}$ and $\mathsf{pk}_{\mathsf{clue}} = \mathsf{pk}$, where $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}$)), and therefore can be linked. Moreover, linkability is inherent to the FMD model: given $\mathsf{pk}_{\mathsf{clue}}$ and $\mathsf{pk}_{\mathsf{detect}}$, one can simple check whether clues generated using $\mathsf{pk}_{\mathsf{clue}}$ are *always* or *rarely* detected using $\mathsf{pk}_{\mathsf{detect}}$.

Both PS schemes [MSS$^+$21] likewise link the detection key to the clue key. In PS2, the model assumes that recipients have a public identifying number R_i serving as the clue key, and this R_i is also (implicitly) provided to the detector in every retrieval by that recipient [MSS$^+$21, Fig. 10, Procedure RECEIVE]. In PS1, there is likewise a persistent identifier R_i known to the receiver at every interaction (and used, e.g., to identify the TEE session eid_i and associated $\vec{L}_i$). The clue key in PS1 is not R_i itself but an RSA-OAEP public key pk_i; yet pk_i is seen by detector and associated with R_i during registration [MSS$^+$21, Fig. 9, procedure Setup, Server line 2].

In both FMD and PS, the detection key and clue key are fixed for each recipient, and therefore the latter linkability issues also hold.

9.2 Defining Key Unlinkability

We can extend the definitions of OMR and OMD, in both the non-DoS (Sections 4.3 and 4.4) and DoS (Section 8.3) variants, to require that the related clue and detection keys are computationally indistinguishable from unrelated ones:

Definition 9.1. An OMR or OMD is *detection-to-clue-key-unlinkable* if for any PPT adversary $\mathcal{A}$, letting $\mathsf{pp} \leftarrow \mathsf{GenParams}(\epsilon_{\mathrm{p}}, \epsilon_{\mathrm{n}})$, $(\mathsf{sk}, \mathsf{pk} = (\mathsf{pk}_{\mathsf{clue}}, \mathsf{pk}_{\mathsf{detect}})) \leftarrow \mathsf{KeyGen}()$ and $(\mathsf{sk}', \mathsf{pk}' = (\mathsf{pk}'_{\mathsf{clue}}, \mathsf{pk}'_{\mathsf{detect}})) \leftarrow \mathsf{KeyGen}()$, it holds that $(\mathsf{pk}_{\mathsf{clue}}, \mathsf{pk}_{\mathsf{detect}}) \equiv_c (\mathsf{pk}_{\mathsf{clue}}, \mathsf{pk}'_{\mathsf{detect}})$.

A stronger notion addresses the latter two linkability issues as well. It lets the recipient generate new clue keys and/or detection keys, that work just as well as the original ones, yet are unlinkable. Defined informally for brevity:

Definition 9.2. An OMR or OMD scheme is *full-key-unlinkable* if it provides two additional algorithms:

- A detection-key regeneration algorithm $\mathsf{pk}^*_{\mathsf{detect}} \leftarrow \mathsf{RegenDetectKey}(\mathsf{sk})$ which a recipient can use to sample a new detection key matching their existing secret key.
- A detection-key regeneration algorithm $\mathsf{pk}^*_{\mathsf{clue}} \leftarrow \mathsf{RegenClueKey}(\mathsf{sk})$ which a recipient can use to sample a new clue key matching their existing secret key.

such that:

- (Soundness and completeness for resampled keys) Messages sent to (re)sampled clue keys are detected by the (re)sampled detection key of the same sk, with error rates bounded analogously to those of the initial $\mathsf{pk}_{\mathsf{clue}}$ and $\mathsf{pk}_{\mathsf{detect}}$.
- (Full key privacy) It is computationally infeasible to distinguish (a) a list of clue keys and detection keys all (re)generated with the same secret key, from (b) a list of freshly-drawn clue keys and detection keys.

9.3 Attaining Unlinkability

Key Unlinkability in OMRp1 and OMRp2. The OMRp1 and OMRp2 schemes are detection-to-clue-key-unlinkable, by the semantic security of BFV (applied to $\mathsf{ct}_{\mathsf{lweSK}}$ in $\mathsf{pk}_{\mathsf{detect}}$). Moreover, they are full-key-unlinkable, via the following key regeneration algorithms.

To resample detection keys, recall that in these schemes, the detection key contains two parts: the BFV public key $\mathsf{pk}_{\mathsf{BFV}}$, which can be resampled from scratch; and the ciphertext $\mathsf{BFV.Enc}(\mathsf{pk}_{\mathsf{BFV}}, \mathsf{PVW.sk})$, which is indistinguishable from $\mathsf{BFV.Enc}(\mathsf{pk}_{\mathsf{BFV}}, 0)$ by semantic security of BFV. Thus, RegenDetectKey is defined as simply re-executing lines 6 to 8 of Algorithm 7. To resample clue keys, recall that in these schemes, the clue key is a PVW public key. PVW encryption naturally supports a RegenPK algorithm that resamples a public key with respect to a given secret key, such that it is indistinguishable from an unrelated public key.[38] Thus, RegenClueKey simply invokes $\mathsf{PVW.RegenPK}(\mathsf{sk})$.

Attaining Key Unlinkability for Other Schemes. Our OMRt1 scheme, as described above, trivially exhibits linkability (since $\mathsf{pk}_{\mathsf{clue}} = \mathsf{pk}_{\mathsf{detect}}$). This can be patched to achieve detection-to-clue-key-unlinkability if the underlying FHE scheme supports resampling public keys, i.e., provides a RegenPK with the aforementioned properties. This is satisfied, e.g., by the aforementioned FHE schemes: FHEW [DM15] and TFHE [CGGI20]. Thus, to achieve detection-to-clue-key-unlinkability for Algorithm 1, change line 7 to define $\mathsf{pk}_{\mathsf{clue}} \leftarrow \mathsf{FHE.RegenPK}(\mathsf{FHE.sk})$. Moreover, full-key-unlinkability can be achieved by defining both RegenDetectKey and RegenClueKey as invoking $\mathsf{FHE.RegenPK}(\mathsf{FHE.sk})$.

PS1 can be modified to achieve detection-to-clue-key unlinkability, as well as RegenClueKey, by changing its PKE to one that (unlike RSA-OAEP) supports resampling public keys, and using this to resample clue keys, analogously to the above. Regenerating detection keys, in this case, requires the recipient to re-register with the server [MSS+21, Fig. 9, procedure Setup] using the same sk (and trust the TEE to not link it to the previous registration that used that sk, which is visible to the TEE). Also, the TEE program [MSS+21, Fig. 8] needs to be modified to not output a linkable pk, and an additional procedure needs to be added to de-register old sessions in order to cease invoking the TEE on these for every new message.

It is not obvious how to achieve full-key-unlinkability, or even the weaker detection-to-clue-key-unlinkablity, for PS2, FMD1, or FMD2.

[38] That is: for this encryption scheme, there exists a PPT public-key-regeneration algorithm RegenPK such that for $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$, $(\mathsf{sk}', \mathsf{pk}') \leftarrow \mathsf{KeyGen}()$ and $\mathsf{pk}^* \leftarrow \mathsf{RegenPK}(\mathsf{sk})$ it holds that $(\mathsf{pk},\mathsf{pk}') \equiv_c (\mathsf{pk},\mathsf{pk}^*)$; and moreover correctness holds with respect to the key pair $(\mathsf{sk},\mathsf{pk}^*)$. In the case of PVW, PVW.RegenPK generates fresh LWE samples from the same distribution as in the original public key.

10 Performance Evaluation

10.1 Methodology

We implemented the $\mathsf{OMRp1}$ scheme of Section 7.3 (and $\mathsf{OMDp1}$ is a simplified version of $\mathsf{OMRp1}$), and the $\mathsf{OMRp2}$ scheme of Section 7.4 instantiated with $\mathsf{SRLC2}$, in a C++ library (released as open source). We used the PALISADE library [PAL21] for PVW encryption, and the SEAL library [Mic20] with Intel-HEXL acceleration [BKS+21] for the BFV scheme. We benchmarked these schemes on several parameter settings, on a Google Compute Cloud `c2-standard-4` instance type (4 hyperthreads of an Intel Xeon 3.10 GHz CPU with 16GB RAM). This section reports the results, in comparison to prior and concurrent works.

Application Parameters. As baseline parameters, we chose the following as representative of a high-traffic cryptocurrency application. The total number of messages is set to $N = 500{,}000$ (roughly the number of Bitcoin payments per day[39]), and the cap on the number of pertinent message per recipient is set to $\bar{k} = 50$ (which, if exceeded, requires repeated retrieval; see Section 7.7). We also show scaling for larger N and $\bar{k}$. We set a false positive rate $\epsilon_{\mathrm{p}} = 2^{-21}$ (including decryption failure) and a false negative rate $\epsilon_{\mathrm{n}} = 2^{-30}$.

The payload size is 612 bytes, as in Zcash (see Section 11).

Internal Parameters. For the PVW encryption, we used PALISADE's implementation [PAL21] with parameters $n = 450$, $q = 65537$, $\sigma = 1.3$, $w = 16000$, $\ell = 4$, for a 120-bits security level according to [APS15, Pla18] and the up-to-date LWE-estimator [CLV+]. For the BFV scheme, we used SEAL [Mic20] with $D = 2^{15}, \log Q = 790, t = 65537$[40], for a security level of > 128 bits according to [APS15, Pla18] and the up-to-date LWE-estimator [CLV+].[41] We set the LWE decryption range $r = 850$, and instantiate with $\mathsf{SRLC2}$ and for simplicity, we fix $\gamma = 5$, $m = 100$ (and thereby directly set γ in $\mathsf{pp}_{\mathsf{SRLC}}$ and m in line 32 in Algorithm 7 and line 40 in Algorithm 8 without calling $\mathsf{GenParams}$), which by empirical tests suffices to achieve $\epsilon_{\mathrm{n}} \ll 2^{-30}$ for $\hat{k} = \lceil \bar{k} + N \log(N)\epsilon_{\mathrm{p}} \rceil = 55$ (empirical tests described in $\mathsf{SRLC2.GenParams.TestRank}$ with $\lambda = 80, \epsilon_{\mathrm{F}} = 2^{-31}$). The resulting multiplicative level is $\sim$23. Since each BFV ciphertext can pack $D \cdot \lfloor \log p \rfloor = 2^{19}$ bits, the digest contains just two BFV ciphertexts: one for the combinations ($m \cdot 612 = 61{,}200$ bytes) and one for the PV vector ($N = 500{,}000$ bits).

For $\mathsf{OMRp2}$ we set two additional parameters (cf. Sections 6.1.2 and 7.4). The number of accumulators for index retrieval is set to $\hat{d} = 1$ (as each BFV ciphertext supports $D = 2^{15}$ slots, so $D > 10\hat{k}$). Each accumulator has two ciphertexts, as we need to represent a number in $N = 500{,}000$, as each ciphertext can represent $2^{16} + 1$ in each slot, so to represent a number $> 2^{16} + 1$ but smaller than $(2^{16} + 1)^2$, we need two ciphertexts. The number of bucket sets is $C = 5$ as discussed in Section 6.1.2, contributing $\ll 2^{-40}$ failure probability to ϵ_{n}. This suffices because the pseudorandom mapping is chosen honestly and independently of the clues. Overall, the $\mathsf{OMRp2}$ digest with our parameter setting is then 16 BFV ciphertexts (including one for payload retrieval).

10.2 Evaluation Results

Representative Costs. Table 3 summarizes the main cost metrics and functionality/security attributes of our scheme, compared to related ones, for the above parameters. (See also Table 1 for

[39]`https://www.blockchain.com/charts/n-payments`, retrieved 2021-08-16

[40]For $\mathsf{OMDp1}$, our circuit has 2 fewer levels, so we use $\log Q = 730$ bits.

[41]$\log Q$ is BFV parameter to support $\sim$23 levels of multiplication.

		Detection schemes				**Retrieval schemes** (including detection)				
		ZIP-307 [GH18, Ele]	PS1 [MSS+21]	PS2 [MSS+21]	OMDp1 §7.2	Zcash full scan [Ele]	FMD1 [BLMG21] / [Lew21b]	FMD2 [BLMG21]	OMRp1 §7.3	OMRp2 §7.4
Communication (bytes/msg)		116	$\ll 1$	$\ll 1$ + 3M s↔s	0.56	612	42	5.3	1.13	9.03
Detector computation time (sec/msg)	1 thread	N/A	0.06	0.25	0.021	N/A	0.011 / 0.00020	0.043	0.145	0.155
	2 threads				0.01				0.075	0.085
	4 threads				0.0099				0.065	0.72
Recipient computation total time (sec)	1 thread	70	$\ll 10^{-3}$	$\ll 10^{-3}$	0.005	61	2.1	0.29	0.02	0.063
Clue size (bytes)		N/A	32	32	956	N/A	68 / 64.5	318,530	956	956
Clue key size (bytes)		N/A	32	N/A	133 k	N/A	1.5 k	1 k	133 k	133 k
Detection key size (bytes)		N/A	64	920	99 M	N/A	768	512	129 M	129 M
Retrieval privacy		Full	Full	Partitioned across detectors	Full	Full	pN-msg-anonymity $p = 2^{-5}$	pN-msg-anonymity $p = 2^{-8}$	Full	Full
Env. assumptions for privacy		None	TEE (SGX)	Non-colluding servers	None	None	None	None	None	None
Env. assumptions for Soundness+completeness		None	Honest S&R	Honest S&R	None	None	Honest S&R	Honest S&R	None	None

Table 3: Comparison of cost metrics, functionality and security attributes. Costs are per message, per recipient. Notation is as in Tables 1 and 2. The bulletin contains N = 500,000 messages, of which $k = \bar{k} = 50$ are pertinent to the recipient. For OMDp1, OMRp1, and OMRp2, we benchmarked 2-thread and 4-thread running time.[42] For FMD1, the detector's computation time is given for both the original implementation [BLMG21] and an optimized re-implementation [Lew21b]. The FMD algorithms in retrieval mode (i.e, for every detected message, the payload is attached.For PS1/PS2, we used the times from [MSS+21, §9.2] (Intel Xeon Platinum 8259CL CPU at 2.5GHz), and some costs are via private communication from their authors. If FMD1/FMD2 are used just for detection, the costs are essentially unchanged except that communication is ≤ 1.

additional functional/security attributes and Table 2 for asymptotic costs,).

We see that in both communication and recipient computation, OMRp1 is better than any other scheme with retrieval functionality, thereby making it attractive for recipients that are limited in bandwidth, computation speed, or energy. Furthermore, OMRp1/OMRp2 provide the strongest form of security, and under the least assumptions, matched only by simple linear scans (full-scan for retrieval, or ZIP-307 for detection) whose communication and recipient computation costs are higher by by orders of magnitude.

Retrieval Scaling with #Messages. Fig. 3 evaluates how the recipient's total cost of retrieval scales with increasing bulletin size N, keeping the number of messages intended for the recipient $\bar{k}$ constant. Only retrieval schemes are included.

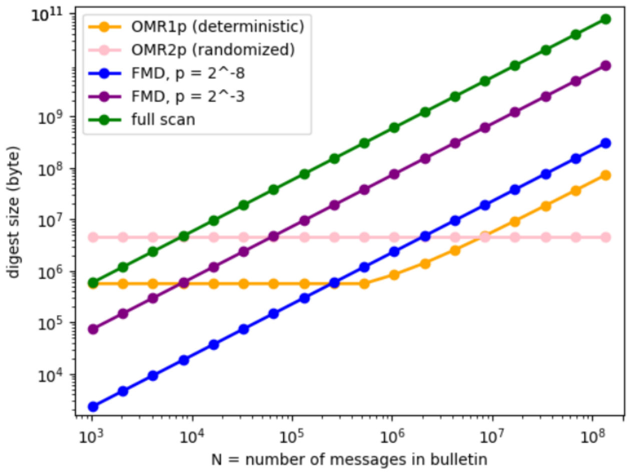
As can be seen, our scheme OMRp1 outperforms all prior constructions starting , in both digest size (for $N \geq 2 \cdot 10^5$) and recipient computation time (for $N \geq 8 \cdot 10^4$).

For $N > 8 \cdot 10^6$, our compact OMR scheme OMRp2 takes the lead and achieves an amortized digest size of **less than 1 bit per message**. In general, the crossover point grows with k (due to the growing number of buckets in OMRp2), so OMRp2 outperforms when N is large but k is small.

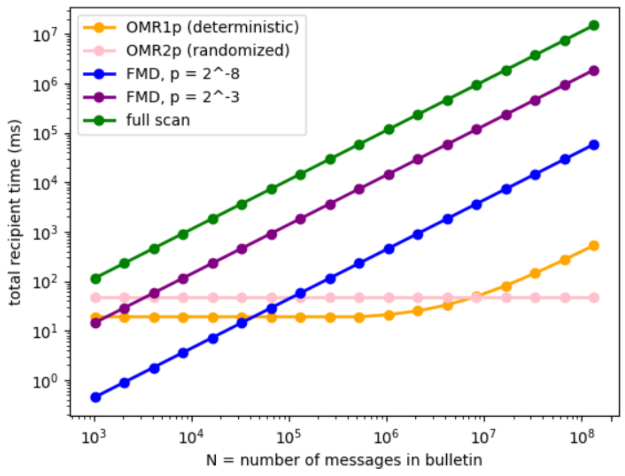
The knee in the OMRp1 running time reflects the transition from running time being dominated by Gaussian elimination (which depends on $\bar{k}$, fixed here) to dominated by decryption (which is linear in N). The knee in the OMRp1 digest size reflects a minimum digest size: a single (packed) BFV ciphertext representing 500,000 messages.

The detector computation time in OMRp1 and OMRp2 is worse than for FMD and (obviously)

[42]We have also run 4-thread benchmarks on GCP instance with the same CPU but with 8 hyperthreads. The result is then 0.063 for OMRp1, and 0.071 for OMRp2. The running time difference is mainly due to that the 4-hyperthread GCP instance typically allocates only two physical cores to our VM.



(a) Digest size vs. number of messages.



(b) Total recipient computation time vs. number of messages.

Figure 3: Retrieval cost comparison (total digest size and recipient computation) for $\bar{k} = 50$.

N	$\bar{k} = 50$		$\bar{k} = 100$		$\bar{k} = 150$	
500,000	549.316	20	823.974	27	1098.632	36
1,000,000	823.975	22	1098.632	29	1647.949	38
2,000,000	1373.291	26	1647.949	33	1922.607	41

Table 4: Digest size (kB) | Recipient running time (ms) in OMRp1 retrieval. N is the number of total messages and $\bar{k}$ is the bound on the number of pertinent messages.

full-scan. It is essentially linear in N for all schemes in this parameter range, and thus follows from Table 3.[43]

OMRp1 dependence on N and $\bar{k}$. Table 4 shows how cost changes when varying both the total number of messages N and the pertinent message bound $\bar{k}$. Only (total) digest size and receiver computation are listed, since these vary with N and $\bar{k}$. Note that as N grows, with constant $\bar{k}$, the amortized per-message digest size shrinks. Due to the packing in the underlying BFV encryption, N is effectively rounded up to a multiple of 500,000; we thus list just such multiples.

For mere *detection* with OMRp1 (omitting its retrieval part), the digest size is 4.5 bits/msg for all of these parameter choices.

Detection Key. The detection key size is 129 MB, using the size-reduction techniques of Section 7.8.[44] This includes the BFV encryption key, the BFV evaluation key, and the BFV encryption of $\mathsf{sk}_{\mathsf{LWE}}$. The BFV encryption key takes 3.3 MB, and likewise the encryption of $\mathsf{sk}_{\mathsf{pvw}}$ (packed into ℓ BFV ciphertext). Size is dominated by the BFV evaluation key, which includes one full-level relinearization key and one full-level rotation key, 50 MB each and some low-level

[43]In particular, for OMRp1 and OMRp2 we fixed $\gamma = 5, m = 2\bar{k}$ which suffices for $\epsilon_n < 2^{-30}$ in all these cases, as well as the PVW and BFV parameters.

[44]Without these optimizations, the detection key size is ∼13.5 GB.

rotation keys totally $\sim$13 MB, for a total of $\sim$129 MB. Note that the detection key does not need to be kept secret; it can be sent to the detector via an insecure channel, authenticated by a simple hash. After the one-time cost of transmission, it can then be used to detect an arbitrary number of messages.

Memory Use. For simplicity, our implementation stores the detection key and all intermediate results in RAM, for a total memory use of $\sim$3.5 GB per thread (i.e., dominated by thread-local variables). [45]

Streaming Finalization Cost. Using the streaming approach of Section 7.5, we can reduce the response time to recipients, by doing most ($> 95\%$) of the detectors' work in advance. For $\mathsf{OMRp1}$, the finalization step (lines 31–39 in Algorithm 7) is only $\sim$0.0021 second per message, single-threaded. It can be further reduced to $\sim$0.00035 second per message, single-threaded, since we have fixed γ (meaning the scalar multiplication of line 53 in Algorithm 7 can be done in advance of finalization). Similar results hold for $\mathsf{OMRp2}$.

11 Integration

We proceed to discuss systems aspect of integrating OMR in real-world applications. For concreteness, we consider integration of $\mathsf{OMRp1}$ or $\mathsf{OMRp2}$ with the Zcash cryptocurrency [HBHW21] to solve the problem of receiver metadata leakage [Hor20] from its light wallet protocol [GH18]. This prospective integration illustrates several hurdles and how they can be resolved. We use the same scheme parameters and benchmark data as in Section 10.

Payload Size. In Zcash, each transaction can contain multiple payments, each addressed to some recipient, and represented as an *output description*. The recipient needs 612 bytes of data to ascertain whether the transaction is addressed to it, decrypt its associated data and human-readable memo, and be able to spend it.[46] These 612 bytes are the message payloads for our OMR. (A single BFV ciphertext in the digest can pack 107 such payloads with the parameters of Section 10.2).

Clue Key Distribution. In our OMR approach (as for FMD [BLMG21] and single-server PS [MSS$^+$21]), senders need to obtain the prospective recipient's clue key in order to generate clues. It is natural to consider the clue key to be an extension of the recipient's public address, shared by the same trusted channels. Zcash's Unified Addresses mechanism [HWH$^+$21] indeed allows such data to be included with public addresses in a backwards-compatible way, and payment URIs [SC12, NH20] can be similarly extended. The clue key size of 133 kB (induced by the PVW public key) has usability issues: it is too large for standard QR codes, or for full display on small screens, although it is still easy to compare such addresses manually.[47]

Alternatively, the Unified Address can contain a short URL from which the clue key can be fetched (and which can point, e.g., to the recipient's secure web server, a Tor hidden service, or

[45]The memory working set can be reduced as follows. Recall that the processing of every message, via homomorphic evaluation, uses the aforementioned evaluation key components in a fixed sequence. We can thus reschedule the execution to process large batches of messages in lockstep, keeping just a couple of currently-needed key components in memory, and swapping them from storage to memory in sequence.

[46]We consider Zcash shielded transactions, and specifically the output descriptions in the Zcash Sapling protocol [HBHW21, §4.5 §4.19 §7.4]. The required fields are the 32-byte ephemeral public key epk and 580-byte ciphertext $\mathsf{C}^{\mathsf{enc}}$ (whose decryption encodes additional values).

[47]E.g., by comparing a truncated prefix, thanks to address hardening [HWH$^+$21].

IPFS). More generally, one can on any general-purpose registry or key-value storage system with suitable privacy and availability guarantees.

Zcash diversified addresses [HBHW21] can be accompanied by different clue keys while preserving address unlinkability, using the full-key-unlinkability property of OMRp1/OMRp2(i.e., the RegenClueKey procedure of Section 9.3). This means that incoming payments, sent to *any* of the user's (perhaps numerous) diversified addresses, can be both retrieved and spent using a single key tuple.

Clue Embedding. Clues of size 956 bytes need to be attached to every payload. This is comparable to the roughly 1.3 kB of data (on average) already placed on-chain per such payment.[48]

It would be the cleanest to extend the transaction format with a dedicated clue field. There are also several methods to embed the information in the existing transaction format. First, we can embed `OP_RETURN` data in the Bitcoin-like scripting language that Zcash supports. Second, we can create dummy output descriptions with 0 monetary value, and populate their memo ciphertext with arbitrary data (580 bytes each). Either way, the sender would packetize and encode each clue into multiple `OP_RETURN` records or memo ciphertext in the same transaction; and the detector would detect these and reassemble them into clues.[49]

It is also possible to propagate the clues on the blockchain's peer-to-peer broadcast network but not on the blockchain ledger, analogously to Bitcoin's SegWit; detectors would record the clues from the peer-to-peer network.

Another alternative, as for clue keys, is to rely on a general-purpose key-value storage system with suitable privacy and availability guarantees, where the key is a transaction ID (or Zcash note commitment), and the value is the corresponding clue(s). Finally, clues can be sent off-chain directly to the detector(s), as in Private Signaling model [MSS+21], though this seems less appropriate for a decentralized blockchain, and creates additional traffic analysis considerations as well as DoS issues (cf. Section 8.2).

Detection Latency. Detectors, in this system, needs to see all blockchain data (as well as peer-to-peer broadcasts, if using the aforementioned SegWit-like approach). Interaction with recipients can happen in a couple of ways.

In the *single-shot* model, the recipient makes a stateless query to the detector: it provides a detection key, a range of blocks to scan, and a bound $\bar{k}$ on the number of pertinent messages; and asks the detector to digest those blocks with respect to that key. The detector runs all of the Retrieve algorithm. Response latency is high: about 0.145 sec per message (cf. Table 3).

The *subscribe and finalize* model utilizes the streaming variant of Section 7.5. The recipient provides a detection key and asks to subscribe to ongoing (and perhaps some past) transactions. The

[48] 948 bytes per payment (output), plus 384 bytes per associated spend (of which there are 0.83 per payment on average), plus 95 bytes transaction header [HBHW21]. That assumes Zcash Canopy transactions, with only Sapling transfers, and less than 253 outputs and spends. The difference between 948 and the aforementioned 612 bytes is due to fields, such as zero-knowledge proofs, which are present on-chain but do not need to be retrieved by a recipient that trusts chain validity.

[49] Alternatively, if using the embedding into dummy output descriptions, the clue fragments can be made indistinguishable from the true ciphertexts of normal output descriptions. This means it would be impossible for anyone (but the recipient) to ascertain whether a transaction even carries clues, or just happens to make many payments. Likewise, this would increase the anonymity set for regular payments. Such indistinguishability is easily achieved: due to hardness of LWE, the clues are indistinguishable from uniformly random $\mathbb{Z}_{2^{16}+1}$ elements, and thus merely needs to be compressed by arithmetic encoding and padded with uniform bits. The detector would then treat any transaction (with sufficiently many output descriptions) as if it carried a clue; and if it doesn't, then OMR DoS-soundness implies not much harm is done.

server starts processing these transactions, doing most of the computation (i.e., homomorphically computing the PV ciphertexts). Later, the recipient shows up and asks the server to finalize the results and pack them into a digest, with respect to some $\bar{k}$. Neither the finalization time nor message bound $\bar{k}$ need be known in advance. This reduces finalization to 0.00035 core-seconds/msg.

Detection Cost. The computational cost for detectors is ∼\$1.02 per million payments scanned (for each recipient served), using commodity cloud computing.[50] This implies \$0.02/month detection cost for Zcash's current shielded payments usage, or \$1.66/month for the current usage rate of Monero (the highest-volume privacy-enhanced cryptocurrency). In the hypothetical case where all of Bitcoin's payments were instead done as Zcash shielded payments, detection cost would grow to \$15.3/month.

Since we deal with a cryptocurrency, the recipient can directly and pay the server for this work, and moreover do so anonymously.

12 Limitations and Future Work

Our results have several limitations and open questions.

Size of Clue and Clue Key. Our 1 kB clue, while comparable to existing system costs, does add significant overhead (in Zcash, it nearly doubles the transaction size). The 133 kB clue size is unwieldy to distribute. Both stem from the underlying LWE-based PVW encryption. Can these be made smaller, while maintaining a practical detection cost?[51]

Detection Cost. The current computational cost of detection, at \$1.02 per million payments scanned, is acceptable for some current blockchains such as Zcash and Monero. However, lower costs are desirable for operation at full Bitcoin scale, or for massive private messaging applications such as Signal or WhatsApp. Acceleration via GPU, FPGA or ASIC can improve costs by orders of magnitude [RCK+21, ABPA+21], and it is likely that further algorithmic improvements are possible in our approach. It may also be possible to amortize detection costs across many recipients.

Detection Key Size. Our BFV-based schemes (OMRp1 and OMRp2) requires recipient to send large detection keys (∼129 MB) to the detectors that serve them. This is a practical hurdle, both in communication (though an insecure channel suffices) and in detector storage. Conversely, OMRt1 instantiated with TFHE reduces detection key size to ∼16 MB, but with much slower detection. Combining the best of the two is an open problem.

DoS-Resistance from Standard Assumptions. Our proof of DoS resistance relies on a new computational hardness conjecture (Conjecture 8.4). Can that conjecture, or a different construction, be proven based on standard assumptions? (Privacy already only on the standard RLWE hardness assumption, even in the DoS threat model.)

Malicious Detectors. Our soundness and completeness properties assume that the detector behaves honestly (though privacy does not). Recipients who doubt that can consult with multiple

[50]At 0.065 seconds/msg, using all 4 vCPUs of a GCP `c2-standard-4` preemptible compute instance billed at \$0.051/hour. For the finalization (0.00035 sec/msg single-core / 0.00018 sec/msg four-core), we assume a non-preemptible instance (\$0.168/hour with sustained use discount) to ensure availability. Communication cost is negligible: $<\$10^{-9}$/msg egress.

[51]Clue size can be losslessly compressed by 6%, since we currently represent $\mathbb{Z}_{2^{16}+1}$ elements using 17 bits, while essentially 16 bits (amortized) would suffice using arithmetic encoding or local base encoding [DPT10]. Clue keys can be shrunk by using Regev05 instead of PVW, but at the cost of enlarging clues. Clues can be shrunk, or even eliminated (cf. Section 6), but at high cost in detection time in our approach.

detectors and hope at least one is honest. Still, it would be preferable to eradicate all honesty assumptions. In principle, zk-SNARK proofs [Mic00, BCCT12] can be used to prove that the detector behaved correctly, but applied generically (on top of an already-expensive FHE computation) the cost would be prohibitive. On the soundness side, application-specific techniques can attest that transactions are verified and mined into a blockchain (e.g., FlyClient [BKLZ20] for Proof-of-Work, and Plumo [GGJ$^+$20] for Proof-of-Stake). Integration of these techniques with OMR, and ensuring completeness for malicious detectors, remain unexplored.

Error Rates. Our schemes have a small but nonzero false-negative rate, due to the LWE-based encryption. Also, our DoS resistance only holds for polynomially-small false-positive rates $\epsilon_{\mathrm{p}} = \mathsf{poly}(\lambda)$, not negligible ones. Practically this does not seem like a great concern, since the parameters of Section 10.2 already achieve false positive rate $\epsilon_{\mathrm{p}} = 2^{-21}$ and false negative rate $\epsilon_{\mathrm{n}} = 2^{-30}$; but it's preferably to reduce these errors further.

Group OMR. Some applications allow messages to be addressed to groups of recipient. These can benefit from a generalization of OMR that allows clues that are detected as pertinent by multiple recipients, ideally without leaking the number of recipients.

Hybrid OMR+FMD. OMR can be combined with FMD [BLMG21] for a flexible cost/privacy tradeoff. For example, first apply FMD with a high decoy rate of $p = 1/4$, to let the detector rule out 75% of the messages; and then apply OMR to the rest for a near-$\times 4$ cost reduction.

Integrations. There remains to realize the integration of our OMR schemes into existing and future systems (e.g., as discussed in Section 11), and to explore the new functional and performance requirements that emerge therefrom.

Acknowledgements

We are grateful to Daniele Micciancio for suggesting suitable FHE schemes for Section 6; to Ran Canetti, Oded Regev and Noah Stephens-Davidowitz for observations on Conjecture 8.4; to Matthew Green, Jack Grigg, Daira Hopwood, Taylor Hornby and Madarz Virza for ideas and observations regarding Zcash integration in Section 11; to István András Seres and Varun Madathil for assistance in quantitative evaluation of [MSS$^+$21] and comparisons to our work; to Miranda Christ for excellent editorial suggestions; and to Wei Dai for assistance in generating level-specific rotation keys using SEAL library.

This material is based upon work supported by DARPA under Contract No. HR001120C00; the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research under award number DE-SC-0001234, the Columbia-IBM center for Blockchain and Data Transparency; and JPMorgan Chase & Co. Any opinions, views, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Government, DARPA, DOE, JPMorgan Chase & Co. or its affiliates, or other sponsors.

References

[ABPA$^+$21] Ahmad Al Badawi, Yuriy Polyakov, Khin Mi Mi Aung, Bharadwaj Veeravalli, and Kurt Rohloff. Implementation and performance evaluation of RNS variants of the bfv homomorphic encryption scheme. *IEEE Transactions on Emerging Topics in Computing*, 2021.

[ACC$^+$18] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Nov 2018.

[ACLS18] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE S&P*, pages 962–979. IEEE Computer Society Press, May 21–23, 2018.

[Ajt96] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *ACM Symposium on Theory of Computing*, STOC '96, page 99–108. ACM, 1996.

[AKS83] M. Ajtai, Janos Komlos, and E. Szemeredi. Sorting in clogn parallel steps. *Combinatorica*, 01 1983.

[ALP$^+$21] Asra Ali, Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Phillipp Schoppmann, Karn Seth, and Kevin Yeo. Communication–computation trade-offs in PIR. In *(USENIX Security 21)*, pages 1811–1828. USENIX, August 2021.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, pages 169–203, 2015.

[AS16] Sebastian Angel and Srinath Setty. Unobservable communication over fully untrusted infrastructure. In *(OSDI 16)*, pages 551–569. USENIX, November 2016.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS'12*, page 326–349. ACM, 2012.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. In *STOC '13*, pages 111–120, 2013.

[BCG$^+$20] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. In *2020 IEEE S&P (SP)*, pages 947–964, 2020.

[BEM$^+$17] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *SOSP*, pages 441–459, 2017.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS 2012*, pages 309–325. ACM, January 8–10, 2012.

[BJT18] Suzie Brown, Oliver Johnson, and Andrea Tassi. Reliability of broadcast communications under sparse random linear network coding. *IEEE Transactions on Vehicular Technology*, 67(5):4677–4682, 2018.

[BKLZ20] Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. Flyclient: Super-light clients for cryptocurrencies. In *2020 IEEE S&P (SP)*, pages 928–946, 2020.

[BKS$^+$21] Fabian Boemer, Sejun Kim, Gelila Seifu, Fillipe DM de Souza, Vinodh Gopal, et al. Intel HEXL (release 1.2). `https://github.com/intel/hexl`, September 2021.

[BLMG21] Gabrielle Beck, Julia Len, Ian Miers, and Matthew Green. Fuzzy message detection. The ACM Conference on Computer and Communications Security (CCS) 2021, 2021.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *CRYPTO 2012*, LNCS, pages 868–886. Springer, Heidelberg, Germany, August 19–23, 2012.

[BSCG$^+$14] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE S&P*, pages 459–474, 2014.

[BSW09] John Bethencourt, Dawn Xiaodong Song, and Brent Waters. New techniques for private stream searching. *ACM Trans. Inf. Syst. Secur.*, 12:16:1–16:32, 2009.

[CGBM15] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE S&P*, pages 321–338, 2015.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, pages 34–91, January 2020.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th FOCS*, pages 41–50. IEEE Computer Society Press, October 23–25, 1995.

[CH18] Hao Chen and Kyoohyung Han. Homomorphic lower digits removal and improved fhe bootstrapping. In *Advances in Cryptology – EUROCRYPT 2018*, Cham, 2018. Springer.

[CLV$^+$] Benjamin Curtis, Cedric Lefebvre, Fernando Virdia, Florian Göpfert, James Owen, Léo Ducas, Markus Schmidt, Martin Albrecht, Rachel Player, and Sam Scott. Security estimates for the learning with errors problem. URL: `https://bitbucket.org/malb/lwe-estimator/src/master/`.

[DD07] George Danezis and Claudia Diaz. Space-efficient private search with applications to rateless codes. In *FC'07*, page 148–162. Springer, 2007.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In *EUROCRYPT 2015, Part I*, LNCS, pages 617–640. Springer, Heidelberg, Germany, April 26–30, 2015.

[DPT10] Yevgeniy Dodis, Mihai Patrascu, and Mikkel Thorup. Changing base without losing space. In *ACM Symposium on Theory of Computing*, STOC '10, page 593–602. ACM, 2010.

[EGNS15] Nitesh Emmadi, Praveen Gauravaram, Harika Narumanchi, and Habeeb Syed. Updates on sorting of fully homomorphic encrypted data. In *2015 ICCCRI*, pages 19–24, 2015.

[Ele] Electric Coin Company. Zcash Rust crates. `https://github.com/zcash/librustzcash`. Commit hash: 99d877e22d58610dc43021b831a28286ef353a89.

[FR13] Matthieu Finiasz and Kannan Ramchandran. Private stream search at almost the same communication cost as a regular search. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography*, pages 372–389, Berlin, Heidelberg, 2013. Springer.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. `https://ia.cr/2012/144`.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *ACM Symposium on Theory of Computing*, STOC '09, page 169–178. ACM, 2009.

[GGJ$^+$20] Ariel Gabizon, Kobi Gurkan, Philipp Jovanovic, Georgios Konstantopoulos, Asa Oines, Marek Olszewski, Michael Straka, Eran Tromer, , and Psi Vesely. Plumo: Towards scalable interoperable blockchains using ultra light validation systems. 3rd ZKProof Standardization Workshop, 2020. URL: `https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-plumo_celolightclient.pdf`.

[GH18] Jack Grigg and Daira Hopwood. Zcash improvement proposal 307: Light client protocol for payment detection. `https://zips.z.cash/zip-0307`, September 2018.

[GLA17] Pablo Garrido, Daniel E. Lucani, and Ramón Agüero. Markov chain model for the decoding probability of sparse network coding. *IEEE Transactions on Communications*, 65(4):1675–1685, 2017.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016.

[Hal05] Shai Halevi. A sufficient condition for key-privacy. Cryptology ePrint Archive, Report 2005/005, 2005.

[HBHW21] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash Protocol Specification Version 2021.2.14. `https://github.com/zcash/zips/blob/master/protocol/protocol.pdf`, 2021.

[HHCP18] Kyoohyung Han, Seungwan Hong, Jung Hee Cheon, and Daejun Park. Efficient logistic regression on large encrypted data. Cryptology ePrint Archive, Report 2018/662, 2018.

[Hor20] Taylor Hornby. Fixing privacy problems in the Zcash light wallet protocol. `https://defuse.ca/downloads/Fixing%20Privacy%20Problems%20in%20the%20Zcash%20Light%20Wallet%20Protocol.pdf`, Oct 2020.

[Hor21] Taylor Hornby. Zip 314 - privacy upgrades to the zcash light client protocol, Mar 2021. URL: `https://forum.zcashcommunity.com/t/zip-314-privacy-upgrades-to-the-zcash-light-client-protocol/38868`.

[HWH$^+$21] Daira Hopwood, Nathan Wilcox, Taylor Hornby, Jack Grigg, Sean Bowe, Kris Nuttycombe, and Ying Tong Lai. Zcash improvement proposal 316: Unified addresses and unified viewing keys. `https://zips.z.cash/zip-0316`, April 2021.

[INZ21] Ilia Iliashenko, Christophe Nègre, and Vincent Zucca. Integer functions suitable for homomorphic encryption over finite fields. Cryptology ePrint Archive, Report 2021/1335, 2021. To appear in WAHC 2021.

[JvdHLZZ15] Jelle Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *SOSP*, page 137–152. ACM, 2015.

[KC16] Amjad Saeed Khan and Ioannis Chatzigeorgiou. Improved bounds on the decoding failure probability of network coding over multi-source multi-relay networks. *IEEE Communications Letters*, 20(10):2035–2038, 2016.

[KDE$^+$21] Andrey Kim, Maxim Deryabin, Jieun Eom, Rakyong Choi, Yongwoo Lee, Whan Ghang, and Donghoon Yoo. General bootstrapping approach for rlwe-based homomorphic encryption. Cryptology ePrint Archive, Report 2021/691, 2021.

[KS07] Tali Kaufman and Madhu Sudan. Sparse random linear codes are locally decodable and testable. In *FOCS'07*, 2007.

[Lai] Kim Laine. Simple encrypted arithmetic library 2.3.1. `https://www.microsoft.com/en-us/research/uploads/prod/2017/11/sealmanual-2-3-1.pdf`. Microsoft Research, Redmond, WA.

[Lew21a] Sarah Jamie Lewis. Discreet log #1: Anonymity, bandwidth and Fuzzytags, Feb 2021. URL: `https://openprivacy.ca/discreet-log/01-anonymity-bandwidth-and-fuzzytags/`.

[Lew21b] Sarah Jamie Lewis. fuzzytags, 2021. URL: `https://git.openprivacy.ca/openprivacy/fuzzytags.git`.

[LMT11] Xiaolin Li, Wai Ho Mow, and Fai-Lung Tsang. Singularity probability analysis for sparse random linear network coding. In *2011 ICC*, pages 1–5, 2011.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 2013.

[LTHA$^+$20] Duc Le, Lizzy Tengana Hurtado, Adil Ahmad, Mohsen Minaei, Byoungyoung Lee, and Aniket Kate. A tale of two trees: One writes, and other reads. *Privacy Enhancing Technologies*, pages 519–536, 04 2020.

[Lun18] Joshua Lund. Technology preview: Sealed sender for signal. `https://signal.org/blog/sealed-sender/`, Oct. 2018.

[LZ16] David Lazar and Nickolai Zeldovich. Alpenhorn: Bootstrapping secure communication without leaking metadata. In *OSDI 16*, pages 571–586. USENIX, November 2016.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, (4):1253–1298, 2000. Preliminary version appeared in FOCS '94.

[Mic20] Microsoft SEAL (release 3.6). `https://github.com/Microsoft/SEAL`, November 2020. Microsoft Research, Redmond, WA.

[MKA$^+$21] Ian Martiny, Gabriel Kaptchuk, Adam Aviv, Dan Roche, and Eric Wustrow. Improving signal's sealed sender. In *NDSS 2022*, 01 2021. NDSS 2022.

[MSS$^+$21] Varun Madathil, Alessandra Scafuro, István András Seres, Omer Shlomovits, and Denis Varlakov. Private signaling. Cryptology ePrint Archive, Report 2021/853 (20210624:145011), 2021.

[MWS$^+$19] Sinisa Matetic, Karl Wüst, Moritz Schneider, Kari Kostiainen, Ghassan Karame, and Srdjan Capkun. BITE: Bitcoin lightweight client privacy using trusted execution. In *(USENIX Security 19)*, pages 783–800. USENIX, August 2019.

[NH20] Kris Nuttycombe and Daira Hopwood. Zcash improvement proposal 321: Payment request URIs. `https://zips.z.cash/zip-0321`, August 2020.

[Noe15] Shen Noether. Ring signature confidential transactions for monero. *IACR Cryptology ePrint Archive*, 2015:1098, 2015.

[OMR21] Oblivious message retrieval implementation. `https://github.com/ZeyuThomasLiu/ObliviousMessageRetrieval`, December 2021.

[OS05] Rafail Ostrovsky and William E. Skeith. Private searching on streaming data. In *CRYPTO*, 2005.

[PAL21] PALISADE lattice cryptography library (release 11.2). `https://palisade-crypto.org/`, June 2021.

[Pla18] Rachel Player. *Parameter selection in lattice-based cryptography.* PhD thesis, Royal Holloway, University of London, 2018.

[PS73] Mike Paterson and Larry Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.*, pages 60–66, 03 1973.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO 2008*, LNCS, pages 554–571. Springer, Heidelberg, Germany, 2008.

[RAD78] R L Rivest, L Adleman, and M L Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation*, pages 169–179, 1978.

[RCK+21] Brandon Reagen, Woo-Seok Choi, Yeongil Ko, Vincent T. Lee, Hsien-Hsin S. Lee, Gu-Yeon Wei, and David Brooks. Cheetah: Optimizing and accelerating homomorphic encryption for private inference. In *2021 IEEE HPCA*, pages 26–39, 2021.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, September 2009.

[SC12] Nils Schneider and Matt Corallo. Bitcoin improvement proposal 21: URI scheme. `https://github.com/bitcoin/bips/blob/master/bip-0021.mediawiki`, January 2012.

[SGGC14] Daniel Salmond, Alex J. Grant, Ian Grivell, and Terence Chan. On the rank of random matrices over finite fields. *CoRR*, 2014. URL: `http://arxiv.org/abs/1404.3250`, `arXiv:1404.3250`.

[SPB21] István András Seres, Balázs Pejó, and Péter Burcsi. The effect of false positives: Why fuzzy message detection leads to fuzzy privacy guarantees? Cryptology ePrint Archive, Report 2021/1180, 2021. `https://ia.cr/2021/1180`.

[SV14] N. Smart and F. Vercauteren. Fully homomorphic simd operations. *Designs, Codes and Cryptography*, 71:57–81, 2014.

[TCL16] Andrea Tassi, Ioannis Chatzigeorgiou, and Daniel Lucani. Analysis and optimization of sparse random linear network coding for reliable multicast services. *IEEE Transactions on Communications*, pages 285–299, 01 2016.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *TCC*, TCC '08, pages 1–18, 2008.

[WCGFJ12] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *OSDI 12*, pages 179–182. USENIX, October 2012.

[WMS+19] Karl Wüst, Sinisa Matetic, Moritz Schneider, Ian Miers, Kari Kostiainen, and Srdjan Capkun. Zlite: Lightweight clients for shielded Zcash transactions using trusted execution. In *Financial Cryptography and Data Security 2019*, LNCS, pages 179–198. Springer, 2019.

A Summary of Constructions

Scheme	Where discussed	Pseudocode	Functionality	Compact	Approach
OMDt1	Section 6.1.1	Algorithm 1	Detection	No	Theoretical (generic FHE)
OMDt2	Section 6.1.2	Algorithm 2	Detection	Yes	Theoretical (generic FHE + bucket-based accumulation)
OMRt1	Section 6.3.2	Algorithm 5	Retrieval	Yes	Theoretical (OMDt2 + SRLC1)
OMDp1	Section 7.2	Simplified Algorithm 7	Detection	No	Practical (BFV + PVW + deterministic index retrieval)
OMRp1	Section 7.3	Algorithm 7	Retrieval	No	Practical (OMDp1 + SRLC2)
OMRp2	Section 7.4	Algorithm 8	Retrieval	Yes	Practical (BFV + PVW + randomized index retrieval + SRLC2)

Table 5: Summary of all the schemes introduced in the paper.

Table 5 provides a summary of all the schemes introduced in this paper.

B Additional Techniques

B.1 Additional Improvements to Compact Detection

The improvements described in this section are imilar to techniques in [DD07]. They are not included in the implementation [OMR21] and reported performance Section 10.

Unit Propagation. A further optimization is to try to resolve collisions by an iterative algorithm that makes a deduction from information already recovered. Every time we recover a new pertinent message from a non-empty collision-free bucket, we can deduce what other buckets that message was mapped to by the PRF f. For each such bucket, we can remove the message from that bucket (by subtracting the message index j from that bucket, and subtracting 1 from the corresponding counter). This may leave some buckets with a single pertinent message, i.e., make them non-colliding, from which a new message can be recovered, and so on. We call this process *unit propagation*. The number of such iterations (until either success or getting stuck) is trivially bounded by $C \cdot m$.

Subset-sum Resolution. Another improvement comes from looking at buckets with a collision (e.g., containing the sum of multiple pertinent message indices), and trying to deduce what set of indices they could contain, as a subset-sum problem.[52] In the general case, this is a computationally hard subset-sum problem; but the constants here are small.

Additionally, *unit propagation* and *subset-sum resolution* can be combined in turn (i.e. first do self resolving and then unit proporgation and then self resolving, and so on until finishes or no new information appears). With our empirical test, for $k = 10$, and $N = 1000$ (self-resolving needs N), $m = 30, C = 2$ is more than enough for failure probability $\ll 2^{-21}$ (we tested for 2^{23} trails and no trail fails,) which is about 1/25 space as the naive method. We leave the failure probability and computational cost bound calculation and deployment of these two techniques for future works. For this work, we stop at gathering partial information.

One thing we would like to note is that these techniques are both introduced in [DD07]. However, they do not have a theoretical bound, so we do not use these techniques directly in our main construction.

[52]For example, suppose message indices {7,12,35} are all mapped by f into the first bucket of the first copy. Suppose moreover that the buckets decrypts to 19 and the corresponding counter decrypts to 2 (i.e., there are two pertinent messages added to the bucket). We can deduce that these are {7,12}.

B.2 Alternative OMD Construction

For the alternative method, we provide OMD with $O(\bar{k}\log(N))$ digests. We do not use it in our main construction because it cannot be extended to our practical construction.

Free-bucket-search Method. We use $\bar{k}$ buckets (each with size $\log(N)$ ciphertexts as above). We also use $\bar{k}$ counters, each encrypting one bit: initially $\langle 0\rangle_{\mathsf{sk}}$, and changed to $\langle 1\rangle_{\mathsf{sk}}$ once one or more pertinent messages have been added to the bucket. For each message m_j, we have $\mathsf{CID}_j \leftarrow \mathsf{PV}_j \cdot j$. Then, we iterate through every bucket Acc_i and try to add CID_j to Acc_i if it is empty (i.e. $\mathsf{Ctr}_i = 0$). This is done homomophically. After addition, since it's based on FHE, we do not know the content of Acc_i. We do homomophic OR gate for all the bits for Acc_i and record the result in Ctr_i (i.e, the counter records whether the bucket is still empty.) Then, if Ctr_i is changed from the $\mathsf{Ctr}_i^{\mathsf{before}}$ before the addition, we zero out CID (done by $\mathsf{CID} \leftarrow \mathsf{CID} \cdot (\mathsf{Ctr}_i \ \mathsf{XOR}\ \mathsf{Ctr}_i^{\mathsf{before}})$). This means that if the counter Ctr_i changed, CID is already correctly recorded in Acc_i and should be zeros for the rest of buckets even though they are also empty. This obviously only have communication cost $O(\bar{k}\log(N))$. The computational cost is then $O(\bar{k}\log(N)N)$ with multiplicative depth $\bar{k}\log(N)$.

Homomorphic Sorting Method. Another way is that we first get all indices $\mathsf{CID} \leftarrow \mathsf{PV}_j \cdot j$, and then use FHE to sort then. Then, we give back the first $\bar{k}$ of then back to the recipient, and also give back the summation of PV to indicate the real k. This also gives us a communication cost of $O(\bar{k} \cdot \log(N))$. The computational cost is then $O(N\log^2(N))$ with multiplicative depth $O(\log^2(N)\log(\ell))$ as shown in [EGNS15] for implemented algorithms, where ℓ in our case is N. If we use AKS sorting network as shown in [AKS83], we may get better results asymptotically like $O(N\log^2(N))$ and $O(\log(N)\log(\ell))$.

B.3 Alternative OMR Construction

The alternative OMR construction is very similar to the alternative construction for OMD. After we get PV's, we multiply PV by (x_i)'s, which result in $\langle x_i\rangle_{\mathsf{sk}}$ or $\langle 0\rangle_{\mathsf{sk}}$. We can then initialize $\bar{k}$ buckets and check if each bucket is empty. Then, if empty, we add the $\mathsf{PV}_i \cdot x_i$ and zero out the result. These can all be done homomorphically as above, so we omit the details. For this construction, we don't need to initialize additional buckets for indices detection, which implies that this construction *does not* imply OMD directly as before. The resulting communication cost is then $O(\bar{k})$ assuming x_i has some constant payload size. The computational cost for the detector is then is then $O(\bar{k} \cdot N)$.