Group Oblivious Message Retrieval

Zeyu Liu[1,*], Eran Tromer[2,*], Yunhao Wang[1,*]

[1]Yale University
[2]Boston University

May 18, 2024

Abstract

Anonymous message delivery, as in private communication and privacy-preserving blockchain applications, ought to protect recipient metadata: a message should not be inadvertently linkable to its destination. But how can messages then be delivered to each recipient, without each recipient scanning all messages? Recent work constructed Oblivious Message Retrieval (OMR) protocols that outsource this job to untrusted servers in a privacy-preserving manner.

We consider the case of group messaging, where each message may have multiple recipients (e.g., in a group chat or blockchain transaction). Direct use of prior OMR protocols in the group setting increases the servers' work linearly in the group size, rendering it prohibitively costly for large groups.

We thus devise new protocols where the servers' cost grows very slowly with the group size, while recipients' cost is low and independent of the group size. Our approach uses Fully Homomorphic Encryption and other lattice-based techniques, building on and improving on prior work. The efficient handling of groups is attained by encoding multiple recipient-specific clues into a single polynomial or multilinear function that can be efficiently evaluated under FHE, and via preprocessing and amortization techniques.

We formally study several variants of Group Oblivious Message Retrieval (GOMR) and describe corresponding GOMR protocols. Our implementation and benchmarks show, for parameters of interest, cost reductions of orders of magnitude compared to prior schemes. For example, the servers' cost is ∼\$3.36 per million messages scanned, where each message may address up to 15 recipients.

[*]Part of the work was done when the three authors were at Columbia University.

Contents

1 Introduction

The protection of message contents in messaging applications is well studied, and end-to-end encryption is nowadays widely practiced. Protecting the metadata, such as sender and recipient identity, is likewise crucial in anonymous message delivery systems [WCGFJ12, CGBM15, Lun18, BEM$^+$17, BLMG], especially when messages are posted on a publicly-visible blockchain [Noe15, BSCG$^+$14, HBHW, BCG$^+$].

Recipient privacy is especially difficult to achieve at scale, because of seemingly contradictory requirements. On one hand, no one should be able to discern the intended recipient of a message (other than that recipient, and the sender). On the other hand, each recipient wishes to retrieve just the messages addressed to them, without scanning all messages in existence for potentially-pertinent ones. There is thus a need to outsource the detection task to a server (or a *detector*) in a privacy-preserving way.

Fuzzy Message Detection [BLMG] was the first to address this, proposing a decoy-based approach in which the server sees the set of messages pertinent to the recipient buried among many additional randomly-chosen messages (which is a weak security notion [SPB22]). Two follow-up works [MSS$^+$22, LT22] improved this to entirely hide the set of pertinent messages, and in [LT22], furthermore ensure unlinkability of keys and resistance to Denial-of-Service or spamming attacks. [LT22] is an essential and practical solution, whose integration is underway by several projects [Con23, Jan22].

Those works focused on the case that a message is sent to a single recipient. What if the sender wants to send a message to multiple recipients, as in group messaging, mailing lists, or blockchain transactions with multiple parties?

Specifically, we consider the following *Group Oblivious Message Retrieval (GOMR)* setting, generalizing [LT22]: a sender sends a message to (up to) G recipients, attaching a *clue* generated using the recipients' *clue keys*. The clue indicates to which recipients this message is addressed. The sender then publishes the message (along with the clue) onto a public *bulletin board*. A *detector*, which is an untrusted third-party server, is enlisted by a recipient: given the recipient's *detection key*, the detector processes the bulletin board and derives a *digest* that contains the pertinent messages, in encrypted form, by an oblivious computation that checks clues against the detection key. The detector sends back the digest to the recipient, who can then extract the messages via their *secret key*.

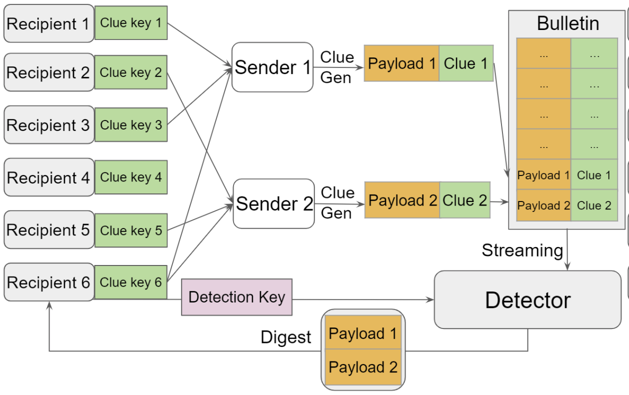
A trivial solution is to utilize prior OMR schemes [LT22], and have each message's sender attach a separate clue for each of the G recipients, inducing a corresponding linear increase in the detector's work — which is impractical for large groups.

In this paper, we construct much more efficient GOMR schemes. We study two models, which differ in how groups are formed (see Fig. 1), motivated by different applications:

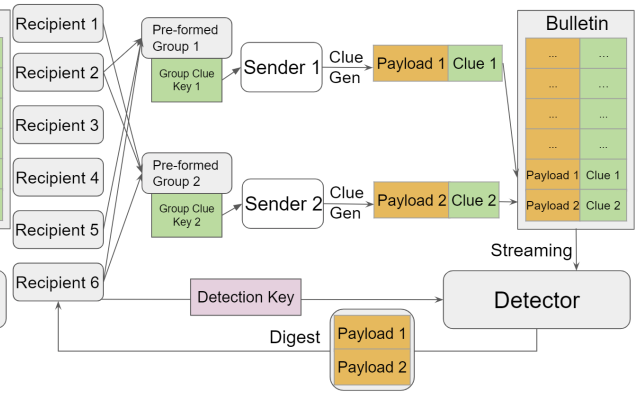
The first flavor is the *Ad-hoc GOMR (AGOMR)*, which allows the senders to send messages to a group of recipients chosen arbitrarily. This suits cases such as messaging protocols (e.g., WhatsApp Broadcast Lists) that let a message be addressed to any set of recipients chosen on the fly, or blockchains where transactions may have many recipients chosen arbitrarily.

The second flavor is *Fixed GOMR (FGOMR)*, where groups are pre-formed by members and addressed collectively. This suits applications with a notion of persistent groups, such as mailing lists or group chats. It also suits blockchains applications in which transactions need to be visible to a set of parties in addition to the recipient (e.g., auditors or jurisdictional law enforcement). FGOMR is a special case of AGOMR, where having pre-formed groups. FGOMR allows for more efficient constructions, and a stronger Denial-of-Service property (two honest recipients cannot be spammed jointly if they do not join the same group).

These two flavors can be extended to include different payloads to different recipients. For example, in the case of Zcash transactions [HBHW], a recipient of funds transaction could retrieve just the output notes



(a) Ad-hoc Group OMR.



(b) Fixed Group OMR.

Figure 1: The high-level illustration of AGOMR and FGOMR. For AGOMR, the sender obtains the individual clue keys to form a group message. For FGOMR, individuals form a group clue key that the sender obtains.

they can spend, instead of the whole transaction (which may include additional notes).

1.1 Our Contribution

Definitions of GOMR. We formally define the two aforementioned models of Group Oblivious Message Retrieval, with fixed groups (FGOMR) and ad-hoc groups (AGOMR).We formally define the aforementioned model of Group Oblivious Message Retrieval for ad-hoc groups.

Our definitions capture natural notions of correctness and privacy for both notions and also include the DoS-resistance and detection-key-unlinkability notions from prior work [LT22] which are extended to the group setting.

A lower bound for GOMR. We prove an information-theoretic lower bound: the clue size for both AGOMR and FGOMR needs to be $\Omega(G)$, where G is the number of members in a group.

Main GOMR constructions. We provide constructions of AGOMR and FGOMR that are far more efficient than applying prior works. The constructions also achieves the strongest functionalities, including the DoS-resistance and detection-key-unlinkability properties introduced in [LT22] (adapted to the group setting).

Our schemes are based on (leveled) homomorphic encryption and achieve privacy guarantees proven under a standard lattice hardness assumption. Our approach encodes multiple recipient-specific clues into a single polynomial or linear function that can be efficiently evaluated under FHE. We utilize preprocessing and amortization of several computational steps and tune homomorphic operation scheduling.

Additional GOMR constructions. In addition to our two main constructionsmain consturction above, we construct fourtwo variants, serving as either a stepping stone to our main constructions or an extension of our main constructions. Compared to our main constructions, most variants sacrifice some properties for efficiency; thus may serve better for other applications. In more detail, if we only accept honestly selected groups instead of arbitrarily formed ones, we can improve the runtime both asymptotically and concretely. Similarly, if we do not require detection-key unlinkability, our schemes can achieve even better detector runtime.

Extension to multi-payload GOMR. Our main AGOMR construction extends to the case where the sender wishes to send different payloads to different recipients (as in OMR), but can batch these together to an ad-hoc group — allowing detection cost to be amortized.

Improving the original OMR scheme. We also improve the single-recipient OMR scheme [LT22] by optimizing procedures and tightening the parameter analysis, resulting in 2.5x detector runtime speedup and 8x shrinkage in digest size.

Lattice-based key-private Multi-Recipient Encryption scheme. As a component of independent interest, we construct the first lattice-based key-private Multi-Recipient Encryption (KP-MRE) scheme. It also serves as a building block for FGOMR constructions.

Implementation and evaluation. We implemented our schemes as an open-sourced C++ library [GOM23] and measured their concrete performances for a variety of parameters in comparison to prior work. With parameters of interest, our schemes can be two to three orders of magnitude faster than the baseline scheme. The FGOMR schemes show better efficiency and scalability than the AGOMR schemes. Concretely, to retrieve half a million messages each addressing 15 recipients, the total cost of only about ∼\$1.68, each message takes ∼0.018 second.

1.2 Related Work

1.2.1 Private Retrieval

Oblivious Message Retrieval. OMR [LT22] addresses the recipient-privacy problem for the case of a single recipient. This work extends this to the group setting. Our privacy, key-unlinkability, and DoS resistance properties are adapted and generalized from [LT22]. An alternative technique to compress the ciphertext is also introduced in [FLS22].

Fuzzy Message Detection. FMD [BLMG] uses a decoy-based privacy notion for single recipients. This is a weaker privacy guarantee [Lew21, SPB21], and does not provide DoS resistance and key unlinkability.

Private Signaling. PS [MSS+22] has the same base privacy guarantee as OMR (i.e., a message cannot be linked to a recipient). However, their constructions rely upon strong environmental assumptions (trusted hardware and two communicating but non-colluding servers, respectively), and their security notion does not provide DoS resistance or key unlinkability. A recent work [JLM23] improves its scalability, but it still assumes trusted hardware. Furthermore, their work does not directly deal with retrieval but just detection. Moreover, as in OMR, their definition is intended to have a message addressed to a single recipient.

PIR. Other related problems are *Private Information Retrieval (PIR)* [CGKS95] and its variant *Keyword PIR [CGN98].*; and in particular, since GOMR recipients retrieve multiple messages, the most related primitive is the variant called *multi-query (keyword) PIR* or *batch (keyword) PIR*. As in OMR, our setting differs in that recipients do not know the indices or labels of messages pertinent to them; rather, the clues are randomized (which is necessary for unlinkability) and require nontrivial computation (rather than simple comparison) to detect. Hence, we utilize general homomorphic encryption, and the resulting costs drive our schemes' design.

Private Stream Search. In Private Stream Search (PSS) [OS05, DD07, BSW09, FR13], a client can search a keyword over a database of documents and retrieve the ones with such a keyword without revealing the keyword to the server. As for Keyword PIR, this does not directly yield OMR or GOMR. In [LT22], the authors use similar techniques as in PSS works. Since our work is built on [LT22] constructions, our schemes also involve those techniques to perform group OMR. However, since those parts are not the main focus of this work, we refer the readers to the original OMR paper for more details.

1.2.2 Multi-Recipient Encryption and Broadcast Encryption

Multi-Recipient Encryption. Multi-Recipient Encryption (MRE) [Kur, BBS02, BBKS07, PPS14] focuses on efficiently encrypting a message (or multiple different messages) to different recipients. In these works, they use the technique of randomness reuse, based on schemes like El Gamal, and achieve an efficiency of 2x compared to the naive one-to-one encryption method. Some recent works [AHK+22, KKPP20, HKP+21] introduce lattice-based schemes. However, these schemes are based on KEM and thus are very inefficient in the setting of OMR as OMR only requires encrypting a small number of bits, while these constructions are only efficient if there are at least λ bits to be encrypted. Furthermore, these schemes are not key-private (see Section 7.2 for more details). We use a novel MRE scheme as a component for FGOMR.

Broadcast Encryption. In Broadcast Encryption (BE) [FN94], a central server controlling a master key is required for distributing keys to clients/recipients. However, in the applications we are interested in, like permissionless blockchain, it is hard to distribute a key from a central server. Moreover, we would also like to protect users' privacy against the central server. Therefore, the BE model is unsuitable for this paper's focus.

[BBKS07] claims that a single-message MRE can also be called broadcast encryption. In this paper, to better distinguish the two different primitives, we only consider the model with a central server as BE, and call the model without a central server MRE.

Messaging Layer Security. Messaging Layer Security (MLS) [ACDT20, ACDT21, HKP22] focuses on secure group messaging. [HKP22] focuses on meta-data hiding of MLS , i.e. hiding the group members of a group. However, its model is different than GOMR and serves different purposes. (1) Functionality: MLS only allows the members inside the group to send to the group, whereas GOMR allows any user in the system to send to any group. This is analogous to symmetric-key vs. public-key encryption. (2) Scaling with the number of groups: MLS requires a user to keep a separate secret key, and perform separate retrieval, for each group they join. In GOMR, each user maintains a single secret key that can be used to fetch all their incoming messages, regardless of how many groups they are in. (Furthermore, in AGOMR, every user is implicitly a member of any ad-hoc group that contains them, which would require an exponential number of retrievals in MLS.) (3) Payload encryption: MLS handles the encryption of payloads. This is out of scope for GOMR (and OMR [LT22]), which assumes the payloads are already suitably encrypted; combining with MLS is the natural way to achieve that. (4) Privacy: GOMR provides stronger metadata hiding, since the server has no idea how many groups even exist, and learns nothing about which group is accessed when. In more detail, in MLS, the server explicitly maintains groups, each with a separate database, and it can observe sending and receiving for each group. MLS reduces the amount of leakage by hiding the group participants (as we do) and can be combined with TOR or VPN to mitigate linkage to IP addresses (with DoS mitigated using signatures [4]), but the number and timing of sending and receiving for each group are still observable and may be exploitable by traffic analysis. Conversely, in GOMR the server maintains a single shared database for all groups, has no idea how many groups even exist, and learns nothing about which group is accessed when.

2 Overview

The following section gives an overview of the model, results, and the main techniques. Section 4 formally defines Ad-hoc Group OMR and Fixed Group OMR. Section 3 covers preliminaries. Section 5 recalls the main protocol of [LT22], and introduces several optimizations later used to build our GOMR constructions. Section 6 and Section 7 introduce our main constructions for AGOMR and FGOMR respectively, and briefly discuss different trade-offs between properties and efficiency which give birth to some variants of our main constructions. Section 8 proposes stronger definitions of GOMR that take DoS attacks into consideration. Section 9 reports on implementation benchmarks and comparisons for the optimized OMR, AGOMR, and FGOMR protocols.

2.1 Model Overview

We first define the following components for GOMR.

A *bulletin board* (or *board* for short), denoted as BB, contains N messages (e.g., blockchain transactions). Each message is sent from a sender, addressed to up to G recipients, whose identities are supposed to remain private. (By contrast, OMR supports only a single recipient.) 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 recipients (in a privacy-preserving way) that the message is addressed to them.

To generate the clue, the sender uses the individual *clue keys* of the intended recipients, or alternatively, a *group clue key* jointly generated by the intended group of recipients. Clue keys, or group clue keys, are assumed to be published or otherwise communicated by some authenticated channels (whose details are outside our scope).

The whole board BB (i.e., all payloads and clues) is public. Typically, payloads will be end-to-end encrypted. Let $\mathcal{P} = \{0,1\}^{\tilde{n}}$ denote the payload space for $\tilde{n} \in \mathbb{Z}^+$, and $\mathcal{C}$ denote the clue space (which depends on the construction).

At any time, potential recipient p may retrieve the messages in BB addressed to groups including p. We denote these messages as *pertinent* (to recipient p), and the rest as *impertinent*.

A server, called a *detector*, helps the recipient *retrieve* the payloads of those pertinent messages. The retrieval is performed obliviously: even a malicious detector learns nothing about which messages are pertinent to the recipient. The recipient gives the detector its *detection key* and a bound $\bar{k}$ on the number of pertinent messages it expects to receive. The detector then aggregates all pertinent messages into a single *digest* string M and sends it to the recipient p. The digest M should be much smaller than the board BB (ideally, proportional to $\bar{k}$).

Assuming the detector is semi-honest and the number of pertinent messages is less than $\bar{k}$, the recipient p should be able to recover all pertinent messages from M with high probability. We denote the probability that a pertinent message is not recovered from the digest as *false negative rate* ϵ_{n} and the probability that the recovery procedure outputs an impertinent message as *false positive rate* ϵ_{p}. Both ϵ_{n} and ϵ_{p} are required to be small (e.g., under 10^{-9} for ϵ_{n} and 10^{-6} for ϵ_{p}).

The board is written incrementally, and the retrieval can be done by specifying the portion of the messages needed.

This work discusses two variants of GOMR: in AGOMR the sender can arbitrarily address any group of (up to G) recipients; whereas in FGOMR, any set of (up to G) recipients can jointly form a group with a corresponding group clue key, and the sender address any of the pre-formed groups. These notions are formally defined in Section 4.

Threat model. We address several attacker goals.

Privacy. We consider an adversary that wishes to learn metadata about which messages are addressed to which user, and about the identity of users that perform message-fetching queries.

The adversary can read all public information (including all board messages and all public keys in the system), and all communication between detectors and the recipients. The adversary may control, or collude with, all the parties in the systems, except for the sender and recipient(s) of the message(s) whose privacy is to be protected. The adversary and its colluding parties may behave maliciously and send malformed messages and keys; but it is computationally bounded (i.e., cannot break the underlying computational assumptions).

Integrity. For completeness and soundness in Definition 4.1 of the retrieved messages, we assume that the adversary is honest-but-curious, but can still collude with detectors and other users. In Section 8, we strengthen our model to allow recipients and senders to be fully malicious, assuming just the detector is honest-but-curious; in particular, we allow the detector to leak its state and the detection keys and queries it observes.

Denial-of-Service resistance. We also consider an adversary that tries to overload recipients by spamming them with messages, in the fully-malicious setting.

2.2 Schemes Summary and Comparisons

Fig. 2 summarizes the asymptotic performance and properties of all the schemes constructed in this paper (see Section 9 for concrete benchmarks). For the comparison baseline (first row), we consider using $\mathsf{OMRp2}$ in OMR [LT22, §7.4] to send separate messages to each group member (see Section 6.1). The two main constructions, alongside the baseline, are shown first. Additional variants are listed below. For reasonable group sizes, all of our schemes are preferred over the baseline; yet they offer various tradeoffs between achieved properties and computation cost, so the choice is application-dependent.

Scheme	Efficiency (per msg)					Properties		
	Clue Size	Detector homomorphic ops			Sender time	Functionality	Key unlinkability	DoS resistance
		Matrix mul size (cheap)	Decrypt (expensive)	Digest coding (expensive)				
OMRp2 [LT22] + Section 6.1	$O(G(n+\ell))$	0	G	G	$O(G(n+\ell)w)$	Arbitrary ad-hoc groups	Full	Ad-hoc DoS
AGOMR3 Theorem 6.4	$O(G(n+\ell))$	$O(G^2\log(P)+ G(n+\ell))$	1	1	$O(G^2(G+n+\ell+\log(P)) +G(n+\ell)w)$	Arbitrary ad-hoc groups	Detection-key unlinkability	Ad-hoc DoS
FGOMR1 Theorem 7.3	$O(n+G\ell)$	$\tilde{O}(G\log(P)\ell)$	1	1	$O(G^2(G\ell+\log(P)) +w(n+\ell G))$	Arbitrary fixed groups and ad-hoc subgroup	Detection-key unlinkability	Fixed DoS
Stepping-stone and variants								
AGOMR1 Theorem 6.1	$O(G(n+\ell))$	0	1	1	$O(G^2(G+n+\ell) +G(n+\ell)w)$	Arbitrary ad-hoc groups	No	Ad-hoc DoS
AGOMR2 Theorem 6.2	$O(G(n+\ell))$	$O(Gn)$	1	1	$O(G^2(G+n+\ell) +G(n+\ell)w)$	Ad-hoc groups formed honestly	Detection-key unlinkability	Ad-hoc DoS
FGOMR2 Theorem 7.4	$O(n+G\ell)$	$O(G\ell)$	1	1	$O(G^3\ell+w(n+\ell G))$	fixed groups formed honestly, arbitrary ad-hoc subgroup	Detection-key unlinkability	Fixed DoS
FGOMR-based AGOMR §7.5	$O(n+G\ell)$	$O(G\ell)$	1	1	$O(G^3\ell+w(n+\ell G))$	ad-hoc groups formed honestly	Detection-key unlinkability	No

Figure 2: The table shows the asymptotic efficiency of our different GOMR constructions in this paper compared to directly using OMR from [LT22] for GOMR. n, ℓ, t, w are PVW encryption parameters. Group size G is the number of recipients in a group. P denotes the total number of recipients in the system. ϵ_{p} denotes the false positive rate when $G = 1$ (i.e., the false positive rate for plain OMR). "Fixed DoS" is stronger than "Ad-hoc DoS". See more details in Section 8. "Arbitrary groups" means that a group can be any subset of recipients, while "groups formed honestly" means that honest senders choose groups with only honest recipients. Detection-key-unlinkability essentially ensures that each retrieval requested by the recipient is unlinkable to the identity of a recipient (cf. Theorem 4.2); full-key-unlinkability also ensures that multiple clue keys under the same secret key are unlinkable.

The detector shoulders the heaviest computation work in the OMR scheme; thus our schemes aim to improve the performance of the detector. As seen in the "Detector homomorphic ops" columns of Fig. 2, the main improvement lies in replacing the expensive homomorphic ciphertext decryption and homomorphic digest coding with relatively cheaper homomorphic matrix multiplications. These multiplications consist of just a single layer of plaintext-by-ciphertext multiplications (or two layers for AGOMR3). By contrast, the homomorphic decryption and digest coding each require multiple levels of ciphertext-by-ciphertext multiplications (each of which is orders of magnitude more expensive than plaintext-by-ciphertext multiplication): homomorphic decryption of PVW encryption scheme [PVW08] involves a range check, which needs to be done using a high-degree polynomial evaluation; and homomorphic digest coding limits the ability to use the SIMD-like batching of the underlying BFV scheme (see details in [LT22]). [1]

Generally, FGOMR schemes have the smallest clue size and a better sender time than the main AGOMR scheme. The sender time of all our schemes is worse than that of the baseline, since the sender proactively performs more computation to alleviate the detector's work (which is more important, as it grows with the *total* number of messages) in our constructions. (Also recall that we prove that for GOMR, the clue size must be $\Omega(G)$ for groups with G recipients.)

Functionality-wise, our major constructions achieve the strongest functionality defined for AGOMR and FGOMR respectively: the groups can be formed arbitrarily. Some of the variants provide better detector performance assuming groups are formed honestly.

For key unlinkability, all schemes (except AGOMR1) achieve detection-key-unlinkability, which means that the detector does not know who the recipient is (in addition to not knowing which messages are pertinent). However, since all schemes require the recipients to include a unique identifier in the clue keys, they do not achieve full unlinkability, which in addition requires that the recipient can generate multiple clue keys that are unlinkable to each other.

DoS resistance for AGOMR is weaker than the one for FGOMR. In FGOMR, it requires that two honest recipients should not be spammed (except with a small probability) as long as they are not in the same

[1] AGOMR1 utilizes an evaluation of a degree-G polynomial over some finite field (e.g., $\mathbb{Z}_{2^{127}-1}$) in plaintext, which is very cheap, to avoid homomorphic matrix multiplications ; the drawback is that AGOMR1 does not provide any detection unlinkability. The other schemes aims to capture detection-key unlinkability, at the cost of using homomorphic matrix multiplication.

group, which is inherently impossible for AGOMR schemes. Details are in Section 8.

For a single payload (i.e., all recipients receive the same payload), all schemes have a similar digest size of $\tilde{O}_\lambda(\tilde{n} \cdot (\bar{k} + N \cdot \epsilon_\mathrm{p}))$ (practically identical digest size for most parameters; see Section 9 for details), where $\tilde{n}$ is the payload size, $\bar{k}$ is the upper bound on the number of pertinent messages, N is the totally number of messages on the board, and ϵ_p is the input false positive rate.

Moreover, all schemes in the table can be extended to take distinct payloads (see Section 6.3), without any change in terms of asymptotic behaviors in the table. The digest size of all schemes also remains the same as before, except for the baseline scheme, where the digest would be G times larger. Thus, our schemes are all better off than the baseline scheme in terms of detector-recipient communication when extended to having distinct payloads.

2.3 Main Techniques

Before proceeding to our main techniques, we recall that in prior OMR [LT22], to send a message to a recipient, the sender first generates a PVW ciphertext. To send a message to G recipients, one simply repeats this process G times. During retrieval, the detector scans all these G messages, thus resulting in the computation work of retrieval processes (including G homomorphic decryption and digest encoding) to increase by a factor of G. This work proposes several techniques to reduce this computation load on the detector.

We briefly summarize the main techniques used in our schemes.

Two optimizations to prior work. We propose two techniques to improve the overall efficiency of the previous OMR constructions: 1) an optimization to $\mathsf{PVUnpack}$ procedure, previously accounting for a large fraction of the detector's runtime, which reduces the number of homomorphic operations from $O(D\log(D))$ to $O(D)$; 2) a tighter parameter analysis by better utilizing the SIMD property of BFV; 3) an approach of compressing messages by concatenating $G > 1$ payloads together to reduce detector cost.

AGOMR using polynomial encoding. To send a message to G recipients (each with some identity ID), the sender first generates G PVW ciphertexts as before. But instead of sending these ciphertexts directly as a clue, the sender instead encodes these into a single polynomial function, by interpolating the polynomial itself at points representing the intended recipients' IDs. This polynomial serves as the clue. Later, the detector evaluates this polynomial at the point corresponding to the retrieving recipient's ID, to recover that recipient's PVW ciphertext. This polynomial evaluation can be done using homomorphic encryption to protect the recipients' identities. After recovering the corresponding ciphertext homomorphically, the detector then only needs to perform a *single* homomorphic decryption and encoding, instead of G such operations for G different clues (as in the baseline method, which again is using OMR naively for GOMR purpose).

Linear encoding and clue size compression. For more efficient homomorphic evaluation, we replace the polynomial encoding by linear encoding. However, naively interpolating a linear function over the IDs of the recipients (which are simply a vector of finite field elements) suspects to attacks: an adversarially-formed group may cause the equation system (for the linear function interpolation) to be unsolvable, breaking completeness. This can be solved by making the ID space sufficiently large (such that the linear equation system is unsolvable with negligible probability), but that increases the clue size. To compress these large IDs, the sender first applies a linear transformation to compress the IDs, where the matrix for the linear transformation is sampled pseudorandomly *after* the group is chosen, and solves the system induced by these compressed IDs.

FGOMR via LWE-based Multi-Recipient Encryption. We construct FGOMR given a key-private Multi-Recipient Encryption (KP-MRE), by instantiating the clues as KP-MRE ciphertexts (instead of PVW ciphertexts).

We thus construct such a KP-MRE based on Learning with Error (LWE), allowing for an FHE-friendly instantiation. The key insight is that given G public keys from G recipients, the sender can construct a single ciphertext $(\vec{a}', b)$ such that for all the corresponding secret keys $(\mathsf{sk}_i)_{i\in[G]}$, it holds that $\langle \vec{a}', \mathsf{sk}_i \rangle \approx b + m$, for a bit m. To do so we divide $\vec{a}'$ and all the secret keys sk_i into two parts, $\vec{a}' = \vec{a}||\alpha$ and $\mathsf{sk}_i = \mathsf{ck}_i||\mathsf{dk}_i$,

$N \in \mathbb{N}$	Size of bulletin board, $\lvert \mathsf{BB} \rvert$
$P \in \mathbb{N}$	Total number of recipients
$\bar{k} \in \mathbb{N}$	Upper bound of the number of messages each recipient has
$\hat{k} \in \mathbb{N}$	Upper bound of the number of messages each recipient has, including false positives
$x \in \mathcal{P}$	Payload
$c \in \mathcal{C}$	Clue
M	Digest
$0 < \epsilon_{\mathrm{p}} < 1$	False positive rate
$0 < \epsilon_{\mathrm{n}} < 1$	False negative rate
$n \in \mathbb{N}$	Dimension of PVW secret key
$w \in \mathbb{N}$	Number of PVW instances in a PVW public key
$\ell \in \mathbb{N}$	Number of bits encrypted in a PVW ciphertext
$t \in \mathbb{N}$	Ciphertext modulus of the underlying PVW scheme; Plaintext modulus of the underlying BFV ciphertext
$\sigma \in \mathbb{R}$	Error standard deviation of the underlying PVW scheme
$D \in \mathbb{N}$	Ring dimension of the underlying BFV ciphertext
$G \in \mathbb{N}$	Maximum number of recipients for each message
$Y \subseteq [P]$	A set of recipients, $\lvert Y \rvert \leq G$
$I \in \mathbb{N}$	Size of ID for each recipient in AGOMR
$\mathsf{id} \in \mathbb{Z}_t^I$	ID of each recipient
$L \in \mathbb{N}$	Size of auxiliary key for each recipient in FGOMR
$\mathsf{dk} \in \mathbb{Z}_t^L$	Auxiliary key of each recipient

Table 1: A list of common parameters used in all the OMR/GOMR constructions.

where dk is a public auxiliary key randomly sampled by the recipients. $\vec{a}, b$ are both computed as in normal LWE ciphertexts (i.e., a subsum of the public key matrices). The sender then computes α such that $\langle \vec{a} || \alpha, \mathsf{sk}_i \rangle \approx b + m$ for all $i \in [G]$.

This construction naturally extends to a multi-bit message $\vec{m}$ and moreover extends to allow each recipient to have their own message (i.e., $\vec{m}_i$ for recipient i).

Generalized snake-eye conjecture. To prove the DoS resistance property of our AGOMR scheme, we generalize the snake-eye conjecture proposed in [LT22]. For FGOMR, we prove it under the same snake-eye conjecture in [LT22]. We also propose another general conjecture and prove it equivalent to the old conjecture in [LT22].

GOMR with distinct payloads. The polynomial encoding technique extends to the case where the recipients in a group each have a different payload. This allows our GOMR schemes to accommodate different payloads for different recipients with a very small overhead.

Amortization and preprocessing. As our schemes highly depend on homomorphic matrix multiplications, we introduce additional techniques that encode the clues and detection keys in special forms (NTT representations) to reduce the cost. The clue encoding and some other homomorphic operations can be performed offline and amortized across multiple recipients.

3 Preliminaries

3.1 Notation

All logarithms are to base 2 and rounded up to an integer, unless otherwise specified. The notation $\vec{v}$ denotes row vectors, and $[n]$ denotes the set $\{1, \ldots, n\}$. $\mathcal{D}(x)$ denotes the distribution of the random variable x.

The main symbols in this paper is summarized in Table 1.

3.2 PVW Encryption

Our constructions are based on lattice-based encryption schemes as in [LT22]. We use the Peikert-Vaikuntanathan-Waters (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. n, w, q, σ are chosen as in [PVW08]. $\mathsf{pp}_{\mathsf{LWE}}$ is assumed to be implicitly taken by the following algorithms.

- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{PVW.KeyGen}(\mathsf{pp}_{\mathsf{LWE}})$: Choose a secret key $\mathsf{sk} \leftarrow \mathbb{Z}_q^{n\times\ell}$ uniformly at random. Randomdly 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 $\mathsf{pk} = (\mathbf{A}, \mathbf{P} = \mathsf{sk}^\mathsf{T}\mathbf{A} + \mathbf{X})$.

- $\mathsf{ct} = (\vec{a}, \vec{b}) \leftarrow \mathsf{PVW.Enc}(\mathsf{pp}_{\mathsf{LWE}}, \mathsf{pk}, \vec{m})$: 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 $(\vec{a}, \vec{b}) = (\mathbf{Ae}, \mathbf{Pe} + \mathbf{t}) \in \mathbb{Z}_q^n \times \mathbb{Z}_q^\ell$.

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

The PVW scheme is unconditionally correct and sound. Under the LWE hardness assumption, it also fulfills the standard definitions of semantic security (IND-CPA) and key privacy [Reg09, PVW08]. Moreover, it satisfies the property of wrong-key decryption defined in [LT22, Definition 5.1].

3.3 Fully Homomorphic Encryption

Fully Homomorphic Encryption (FHE), introduced by Rivest et al. [RAD78] and first constructed by Gentry [Gen09], enables evaluation of a circuit on encrypted data, such that the result is the encryption of the corresponding output.

BFV FHE scheme. We use the Brakerski/Fan-Vercauteran (BFV) homomorphic encryption scheme [Bra12, FV12].

BFV scheme consists of the following PPT algorithms: $\mathsf{GenParams}(1^\lambda), \mathsf{KeyGen}(\mathsf{pp}_{\mathsf{BFV}}), \mathsf{Enc}(\mathsf{pp}_{\mathsf{BFV}}, \mathsf{pk}, m)$, $\mathsf{Dec}(\mathsf{pp}_{\mathsf{BFV}}, \mathsf{sk}, c)$ as normal PKE schemes. And an additional algorithm: $\mathsf{Eval}(\mathsf{pp}_{\mathsf{BFV}}, \mathsf{pk}, (\mathsf{ct}_1, \ldots, \mathsf{ct}_k), C)$, which takes k ciphertexts encrypting $(m_{i\in[k]})$ and a circuit C, and outputs another ciphertext ct' such that $\mathsf{Dec}(\mathsf{pp}_{\mathsf{BFV}}, \mathsf{sk}, \mathsf{ct}') = C(\mathsf{Dec}(\mathsf{pp}_{\mathsf{BFV}}, \mathsf{sk}, m_i)_{i\in[k]})$.

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

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 is from a larger cyclotomic ring $R_q = \mathbb{Z}_q[X]/(X^D + 1)$ for some $q > t$. t, q, and D are the plaintext modulus, the ciphertext modulus, and the ring dimension, respectively. Each ciphertext can pack D plaintext group elements $(m_1, \ldots, m_D) \in \mathbb{Z}_t^D$. We can thus perform "single instruction, multiple data" (SIMD) homomorphic evaluation to BFV ciphertexts.

4 Group Oblivious Message Retrieval

In this section, we formally define the problem of Group Oblivious Message Retrieval, using the model and notation of Section 2.1.

4.1 Ad-hoc Group OMR

Ad-hoc Group OMR addresses the case where senders arbitrarily choose up to G recipients, given the recipients' clue keys. No advance action or cooperation is necessary from the recipients other than publishing clue keys (as in standard OMR). For privacy, an adversary who corrupts the detector should not tell which group a message is addressed to. Furthermore, even if it has corrupted some of the group members (including adaptively corrupting some group members, obtaining their secret keys, and even *maliciously crafting their clue keys*), the adversary should not learn who the *remaining* recipients are.

This definition generalizes that of OMR [LT22, §4.3]; see Theorem 4.1 for detail.

Definition 4.1 (Ad-hoc Group Oblivious Message Retrieval (AGOMR))**.** An AGOMR scheme has the following PPT algorithms:

- $\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, \epsilon_{\mathrm{p}}, \epsilon_{\mathrm{n}}, G, P)$: takes a security parameter λ, false positive rate ϵ_{p}, false negative rate ϵ_{n}, upperbound of the number of recipients G per message, total number of recipients P; output public parameter pp.
- $(\mathsf{sk}, \mathsf{pk} = (\mathsf{pk}_{\mathsf{clue}}, \mathsf{pk}_{\mathsf{det}})) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$: takes public parameters pp; outputs a secret key sk and public key pk consisting of a clue key $\mathsf{pk}_{\mathsf{clue}}$ and detection key $\mathsf{pk}_{\mathsf{det}}$.
- $c \leftarrow \mathsf{GenClue}(\mathsf{pp}, \mathsf{pk}_{\mathsf{clue}1}, \ldots, \mathsf{pk}_{\mathsf{clue}G}, x)$: takes a public parameter pp, up to G clue keys and a payload $x \in \mathcal{P}$; outputs a clue $c \in \mathcal{C}$.
- $M \leftarrow \mathsf{Retrieve}(\mathsf{pp}, \mathsf{BB}, \mathsf{pk}_{\mathsf{det}}, \bar{k})$: takes a public parameter pp, a board $\mathsf{BB} = \{(x_1, c_1), \ldots, (x_N, c_N)\}$ of size N, a detection key $\mathsf{pk}_{\mathsf{det}}$ from the recipient, an upper bound $\bar{k}$ on the number of pertinent messages for that recipient; outputs a digest M.
- $\mathsf{PL} \leftarrow \mathsf{Decode}(\mathsf{pp}, M, \mathsf{sk})$: takes a public parameter pp, the digest M, a secret key sk, and outputs either a decoded payload list $\mathsf{PL} \in \mathcal{P}^k$ or an overflow indication $\mathsf{PL} = \mathsf{overflow}$.

An AGOMR scheme should fulfill completeness, soundness, and computational privacy, as defined below. These definitions use the notion of a board generation procedure:

Definition 4.2 (AGOMR board generation)**.** Given the total number of messages N, the total number of recipients P and a public parameter pp: for each recipient $j \in [P]$, generate keys $(\mathsf{sk}_j, \mathsf{pk}_j) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$; for $i \in [N]$, arbitrarily choose a set $Y_i \subseteq [P]$, $1 \leq |Y_i| \leq G$, representing the set of recipients for the i-th message. Define sets $S_1, \ldots, S_P$ such that $S_j = \{i \mid j \in Y_i\}$ representing the indices of messages addressed to recipient j. Arbitrarily choose unique payloads $(x_1, \ldots, x_N)$. Generate clues $c_i \leftarrow \mathsf{GenClue}(\mathsf{pp}, \{\mathsf{pk}_{\mathsf{clue}j}\}_{j \in Y_i}, x_i)$ for $i \in [N]$.[2] Then, output the set S_1, $(\mathsf{sk}_1, \mathsf{pk}_{\mathsf{det}1})$, and the board $\mathsf{BB} = \{(x_1, c_1), \ldots, (x_N, c_N)\}$.[3]

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

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

  The randomness is over the coins of KeyGen and $\mathsf{GenClue}$.

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

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

- (Computational privacy) An AGOMR scheme is computationally private if there does not exist any PPT adversary that can win the game in Fig. 3 with probability $> 1/2 + \mathsf{negl}(\lambda)$, where Z, Y_0, Y_1 represent groups with at most G recipients.

[2] Payloads being unique is w.l.o.g. See more details in paragraph "Repeating Payloads" in [LT22, Section 4.3]

[3] S_1 is the set containing indices of messages pertinent to the recipient holding keys $\mathsf{sk}_1, \mathsf{pk}_1$, which w.l.o.g is the first recipient. All the properties defined below symmetrically applies to other recipients.

Adversary		**Challenger**
		$\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n}, G, P)$
Choose $Y_0, Y_1 \subseteq [P]$,		For $i \in [P]$:
where $\lvert Y_0\rvert, \lvert Y_1\rvert \leq G$	$\xleftarrow{(\mathsf{pk}_i)_{i\in[P]}, \mathsf{pp}}$	$(\mathsf{sk}_i, \mathsf{pk}_i = (\mathsf{pk}_{\mathsf{clue}\,i}, \cdot)) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$
Let $Z = Y_0 \cap Y_1$	$\xrightarrow{Y_0, Y_1, Z}$	
Choose $(\mathsf{sk}_i, \mathsf{pk}_i)_{i\in Z'}$, where $Z' \subseteq Z$,	$\xleftarrow{(\mathsf{sk}_i)_{i\in Z}}$	
overwriting their old values		
Choose a payload x	$\xrightarrow{Z', x, (\mathsf{sk}_i, \mathsf{pk}_i)_{i\in Z'}}$	$b \leftarrow\$ \{0, 1\}$
	$\xleftarrow{c}$	$c \leftarrow \mathsf{GenClue}(\mathsf{pp}, (\mathsf{pk}_{\mathsf{clue}\,i})_{i\in Y_b}, x)$
	$\xrightarrow{b'}$	(Adversary wins if $b = b'$)

Figure 3: Computational Privacy game for AGOMR

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

- (Compactness) For $\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n}, G, P)$, $(\mathsf{sk}, \mathsf{pk} = (\mathsf{pk}_{\mathsf{clue}}, \mathsf{pk}_{\mathsf{det}})) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, for any board $\mathsf{BB} = \{(x_1, c_1), \ldots, (x_N, c_N)\}$, for $M \leftarrow \mathsf{Retrieve}(\mathsf{pp}, \mathsf{BB}, \mathsf{pk}_{\mathsf{det}}, \bar{k})$, it holds that:

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

The compactness definition is the same as [LT22] (except for some minor interface changes and the $\log(G)$ factor).

$\tilde{O}(\bar{k} + \epsilon_\mathrm{p} N)$ (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.

Remark 4.1 (Relation to OMR)**.** When $G = 1$, this AGOMR definition implies OMR ([LT22, Def 4.1]), with a minor difference in privacy. The adversary in AGOMR definition is given P honestly generated recipient keys and can arbitrarily choose two of them as the challenge, P as the total number of recipients in the system passed as a parameter in $\mathsf{GenParams}$. In contrast, the adversary in [LT22, §4.3] is given two honestly generated keys as the challenge. Other parts are trivially identical.

Detection-key unlinkability. In addition to hiding the results of a retrieval query, we may wish to hide which recipient is even doing the retrieval. Thus, essentially, we require the property that given a detection key of a recipient, one cannot tell who the recipient is.

To satisfy this property, we need the recipient to be able to regenerate its detection key to do the retrieval, while still having the correctness, soundness, privacy, and compactness hold, using the new detection key. Furthermore, the new detection key should not be linkable to the recipient even given the original set of public keys.

We formally capture this property by the following definition, adapted from [LT22, Def 9.2].

Definition 4.3. (AGOMR detection-key-unlinkability) An AGOMR scheme is said to be detection-key-unlinkable if it further has an interface $(\mathsf{sk}', \mathsf{pk}_{\mathsf{det}}) \leftarrow \mathsf{RegenDetectKey}(\mathsf{pp}, \mathsf{sk})$, such that:

1. Correctness, soundness, computational privacy, and compactness hold also after replacing $M \leftarrow \mathsf{Retrieve}(\mathsf{pp}, \mathsf{BB}, \mathsf{pk}_{\mathsf{det}}, \bar{k})$ with $M \leftarrow \mathsf{Retrieve}(\mathsf{pp}, \mathsf{BB}, \mathsf{pk}'_{\mathsf{det}}, \bar{k})$, and replacing $\mathsf{PL} \leftarrow \mathsf{Decode}(\mathsf{pp}, M, \mathsf{sk})$ with $\mathsf{PL} \leftarrow \mathsf{Decode}(\mathsf{pp}, M, \mathsf{sk}')$, where $(\mathsf{sk}', \mathsf{pk}'_{\mathsf{det}}) \leftarrow \mathsf{RegenDetectKey}(\mathsf{pp}, \mathsf{sk})$.

2. Let $\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n}, G, P)$, $(\mathsf{sk}, \mathsf{pk} = (\mathsf{pk}_{\mathsf{clue}}, \mathsf{pk}_{\mathsf{det}})) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, $(\mathsf{sk}', \mathsf{pk}' = (\mathsf{pk}'_{\mathsf{clue}}, \mathsf{pk}'_{\mathsf{det}})) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, then for any $n = \mathsf{poly}(\lambda)$, for all $i \in [n]$, let $(\cdot, \mathsf{pk}_{\mathsf{det}\,i}) \leftarrow \mathsf{RegenDetectKey}(\mathsf{pp}, \mathsf{sk})$, $(\cdot, \mathsf{pk}'_i = (\cdot, \mathsf{pk}'_{\mathsf{det}\,i})) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, it holds that $(\mathsf{pk}, \mathsf{pk}_{\mathsf{det}\,1}, \ldots, \mathsf{pk}_{\mathsf{det}\,n}) \approx_c (\mathsf{pk}', \mathsf{pk}'_{\mathsf{det}\,1}, \ldots, \mathsf{pk}'_{\mathsf{det}\,n})$

Remark 4.2. This *detection-key unlinkability* directly implies *detection-to-clue-key-unlinkablity* [LT22, Def 9.1], but is weaker than *full-key-unlinkablity* [LT22, Def 9.2]. The difference from the latter is that *detection-key unlinkability* does not achieve the clue-key-to-clue-key unlinkability, i.e., the ability for recipients to generate multiple clue keys that are indistinguishable but functionally identical, which is useful in some applications (e.g., ephemeral/stealth addresses). Note that as in [LT22], all key unlinkabilities assume a proper use of network-level mitigation (e.g., using Tor).

4.2 Fixed Group OMR

In some applications, groups are predetermined by their member recipients (e.g., for mailing lists). A recipient joining multiple groups should be able to use a single secret key to detect all pertinent messages addressed to the groups, which include that recipient as a group member. By sending to a fixed group, efficiency can be further improved, including the clue size, server computation time, and so on. (For simplicity, a sending to a single recipient will be handled as sending to a a fixed group consisting of just that recipient.)

Besides the difference in efficiency, FGOMR has a slightly weaker privacy notion. If an adversary corrupts some members in a fixed group, it may distinguish messages addressed to that group from those addressed to other groups (which in reality is often possible anyway by inspecting the payload's plaintext). Thus, we require for messages addressed to groups *without* corrupted recipients, the groups are indistinguishable.

Furthermore, a sender is allowed to select an arbitrary subgroup Y' of some fixed group Y and send a message only pertinent to this subgroup. The excluded recipients $Y \setminus Y'$ should not discover such an exclusion (i.e., the clue generated for the subgroup is indistinguishable from a clue generated for a group that does not include the excluded recipients at all).

The notion of an FGOMR scheme is captured as follows. KeyGen in OMR is replaced with $\mathsf{PersonalKeyGen}$, which only generates secret and detection keys. A clue key for an entire group of up to G recipients will be constructed by first invoking $\mathsf{GroupKeyGenAux}$ to generate a share with respect to that group, and then invoking $\mathsf{GroupKeyGen}$.

Definition 4.4 (Fixed Group Oblivious Message Retrieval (FGOMR))**.** An FGOMR scheme has the following PPT algorithms:

- $\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n}, G, P)$: takes a security parameter λ, a false positive rate ϵ_p, a false negative rate ϵ_n, the upper bound of the number of recipients of each message G, and the total number of recipient P; outputs public parameters pp.
- $(\mathsf{sk}, \mathsf{pk}_\mathsf{det}) \leftarrow \mathsf{PersonalKeyGen}(\mathsf{pp})$: takes public parameters pp; outputs a secret key sk and a detection key pk_det.
- $\mathsf{gPKshare} \leftarrow \mathsf{GroupKeyGenAux}(\mathsf{pp}, \mathsf{sk}, Y)$: takes public parameters pp, a secret key sk, and a group of recipients Y; outputs a group key share $\mathsf{gPKshare}$ which allows the owner of sk to be included in the group Y.
- $\mathsf{pk}_\mathsf{clue} \leftarrow \mathsf{GroupKeyGen}(\mathsf{pp}, \mathsf{gPKshare}_1, \dots, \mathsf{gPKshare}_G)$: takes public parameters pp, up to G public key shares $\mathsf{gPKshare}_i$; outputs a clue key.
- $c \leftarrow \mathsf{GenClue}(\mathsf{pp}, \mathsf{pk}_\mathsf{clue}, Y, x)$: takes public parameters pp, a clue key $\mathsf{pk}_\mathsf{clue}$, a group of recipients Y, a payload $x \in \mathcal{P}$; outputs a clue $c \in \mathcal{C}$.
- $M \leftarrow \mathsf{Retrieve}(\mathsf{pp}, \mathsf{BB}, \mathsf{pk}_\mathsf{det}, \bar{k})$: takes public parameters pp, a board $\mathsf{BB} = \{(x_1, c_1), \dots, (x_N, c_N)\}$ of size N, a detection key pk_det, and an upper bound $\bar{k}$ on the number of pertinent messages addressed to that recipient; outputs a digest M.
- $\mathsf{PL} \leftarrow \mathsf{Decode}(\mathsf{pp}, M, \mathsf{sk})$: takes public parameters pp, a digest M and a secret key sk; outputs either a decoded payload list $\mathsf{PL} \in \mathcal{P}^k$ or an overflow indication $\mathsf{PL} = \mathsf{overflow}$.

A FGOMR scheme should fulfill completeness, soundness, and computational privacy, as defined below. These definitions use a different notion of board generation:

Definition 4.5 (FGOMR board generation)**.** Given the total number of messages N, the total number of recipients P and public parameter pp: for each recipient $j \in [P]$, generate keys $(\mathsf{sk}_j, \mathsf{pk}_{\mathsf{det}j}) \leftarrow \mathsf{PersonalKeyGen}(\mathsf{pp})$; for $i \in [N]$, arbitrarily choose $Y_i \subseteq [P], 1 \leq |Y_i| \leq G$, where Y_i represents the group of recipients for the i-th message, and $Y_i' \subseteq Y_i, Y_i' \neq \phi$. Define sets $S_1, \ldots, S_P$ such that $S_j = \{i \mid j \in Y_i'\}$, representing the indices of messages addressed to recipient j. For all $i \in [N], j \in Y_i$, generate $\mathsf{gPKshare}_{i,j} \leftarrow \mathsf{GroupKeyGenAux}(\mathsf{pp}, \mathsf{sk}_j, Y_i)$, and $\mathsf{pk}_{\mathsf{clue}i} \leftarrow \mathsf{GroupKeyGen}(\mathsf{pp}, \{\mathsf{gPKshare}_{i,j}\}_{j \in Y_i})$. Arbitrarily choose unique payloads $(x_1, \ldots, x_N)$; generate clues $c_i \leftarrow \mathsf{GenClue}(\mathsf{pp}, \mathsf{pk}_{\mathsf{clue}i}, Y_i', x_i)$. Finally, output S_1, $(\mathsf{sk}_1, \mathsf{pk}_{\mathsf{det}1})$, and the board $\mathsf{BB} = \{(x_1, c_1), \ldots, (x_N, c_N)\}$.

- (Completeness and Soundness) Same as AGOMR completeness and soundness in Definition 4.1, replacing Definition 4.2 with Definition 4.5.
- (Computational privacy) An FGOMR scheme is computationally private if there does not exist any PPT adversary that can win the game in Fig. 4 with probability $> 1/2 + \mathsf{negl}(\lambda)$, where Y_j, Z_j represent groups of recipients.

A FGOMR scheme is *compact* if it further satisfies the compactness definition in Theorem 6.1 (w.r.t. the interfaces defined above).

Detection-key unlinkability. Detection-key unlinkability is similarly defined for FGOMR.

Remark 4.3. Note that our definition considers at most two phases ($\mathsf{GroupKeyGenAux}$ followed by $\mathsf{GroupKeyGen}$). This could be generalized to multi-round MPC, which is conceptually straightforward but hard to capture without extra formal machinery. We thus do not include that version of definition and left it to future works that have constructions requiring such extra rounds.

Definition 4.6. (FGOMR detection-key-unlinkability) An FGOMR scheme is said to be detection-key-unlinkable if it further has an interface $(\mathsf{sk}', \mathsf{pk}'_{\mathsf{det}}) \leftarrow \mathsf{RegenDetectKey}(\mathsf{pp}, \mathsf{sk})$, such that:

1. Correctness, soundness, computational privacy, and compactness hold also after replacing $M \leftarrow \mathsf{Retrieve}(\mathsf{pp}, \mathsf{BB}, \mathsf{pk}_{\mathsf{det}}, \bar{k})$ with $M \leftarrow \mathsf{Retrieve}(\mathsf{pp}, \mathsf{BB}, \mathsf{pk}'_{\mathsf{det}}, \bar{k})$, and replacing $\mathsf{PL} \leftarrow \mathsf{Decode}(\mathsf{pp}, M, \mathsf{sk})$ with $\mathsf{PL} \leftarrow \mathsf{Decode}(\mathsf{pp}, M, \mathsf{sk}')$, where $(\mathsf{sk}', \mathsf{pk}'_{\mathsf{det}}) \leftarrow \mathsf{RegenDetectKey}(\mathsf{pp}, \mathsf{sk})$.
2. Let $\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, \epsilon_{\mathrm{p}}, \epsilon_{\mathrm{n}}, G, P)$, $(\mathsf{sk}, \mathsf{pk}_{\mathsf{det}}) \leftarrow \mathsf{PersonalKeyGen}(\mathsf{pp})$, $(\mathsf{sk}', \mathsf{pk}'_{\mathsf{det}}) \leftarrow \mathsf{PersonalKeyGen}(\mathsf{pp})$, for all $Y, Y' \subseteq [P]$, let $\mathsf{gPKshare} \leftarrow (\mathsf{pp}, \mathsf{sk}, Y)$, $\mathsf{gPKshare}' \leftarrow (\mathsf{pp}, \mathsf{sk}, Y')$, for any $n = \mathsf{poly}(\lambda)$, for all $i \in [n]$, let $(\cdot, \mathsf{pk}_{\mathsf{det}i}) \leftarrow \mathsf{RegenDetectKey}(\mathsf{pp}, \mathsf{sk})$, $(\cdot, \mathsf{pk}_i' = (\cdot, \mathsf{pk}'_{\mathsf{det}i})) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, it holds that $(\mathsf{gPKshare}, \mathsf{pk}_{\mathsf{det}}, \mathsf{pk}_{\mathsf{det}1}, \ldots, \mathsf{pk}_{\mathsf{det}n}) \approx_c (\mathsf{gPKshare}, \mathsf{pk}'_{\mathsf{det}}, \mathsf{pk}_{\mathsf{det}1}', \ldots, \mathsf{pk}_{\mathsf{det}n}')$

Remark 4.4. A stronger notion of FGOMR computational privacy is possible, where privacy holds even if the keys are chosen semi-maliciously (i.e., with malicious choice of randomness). This is captured by the game of Fig. 11, which allows the adversary to rewrite some secret keys with its own maliciously-chosen randomness.

An even stronger notion of privacy allows the keys to be crafted fully maliciously (as in AGOMR). This reduces to the above semi-malicious notion, by adding a non-interactive zero-knowledge (NIZK) proof that the keys were correctly generated from some randomness.

Neither strenghtening is essential for the envisioned application, so for readability we omit these from subsequent discussion. Note, however, that all of our FGOMR constructions satisfy the semi-malicious strenghtening, and thus can be made private even for fully-malicious key generation by addition of the NIZK proof to the public keys.

Adversary		Challenger
	$\xleftarrow{\mathsf{pp}}$	$\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n}, G, P)$
		$(\mathsf{sk}_j, \cdot) \leftarrow \mathsf{PersonalKeyGen}(\mathsf{pp})$ for $j \in [P]$
Query Phase I: for $i = 1, \cdots, N$		
Choose recipient set $Y_i \subseteq [P]$, where $\lvert Y_i \rvert \leq G$	$\xrightarrow{Y_i}$	
	$\xleftarrow{(\mathsf{gPKshare}_{j,i})_{j\in Y_i}}$	$\mathsf{gPKshare}_{j,i} \leftarrow \mathsf{GroupKeyGenAux}(\mathsf{pp}, \mathsf{sk}_j, Y_i)$, for $j \in Y_i$
Challenge Phase:		
Choose $i_0 \neq i_1 \in [N]$, and for $i \in [i_0, i_1]$:		
Choose $Z_i \subseteq Y_i$	$\xrightarrow{Z_i}$	
Choose $(\mathsf{gPKshare}'_{j,i})_{j\in Z_i}$,	$\xleftarrow{(\mathsf{sk}_j)_{j\in Z_i}}$	
and $\mathsf{gPKshare}'_{j,i} = \mathsf{gPKshare}_{j,i}, j \in Y_i \setminus Z_i$		
Choose a payload x	$\xrightarrow[((\mathsf{gPKshare}'_{j,i})_{j\in Y_i})_{i\in[i_0,i_1]}]{i_0, i_1, x, Z_0, Z_1}$	$b \leftarrow\$ \{0,1\}$
		$\mathsf{pk}_{\mathsf{clue}_{i_b}} \leftarrow \mathsf{GroupKeyGen}(\mathsf{pp}, (\mathsf{gPKshare}'_{j,i_b})_{j\in Y_{i_b}})$
	$\xleftarrow{c}$	$c \leftarrow \mathsf{GenClue}(\mathsf{pp}, \mathsf{pk}_{\mathsf{clue}_{i_b}}, Y_{i_b} \setminus Z_{i_b}, x)$
Query Phase II: same as **Query Phase I**.	$\xrightarrow{b'}$	(Adversary wins if $b = b'$)

Figure 4: Computational Privacy game for FGOMR

4.3 Lower Bounds

Lower bound for AGOMR. This model implies that the size of clues must be at least linear in the group size. Formally:

Theorem 4.5. For any AGOMR scheme AG, for any $\lambda > 0$, $0 < \epsilon_\mathrm{n} < 1 - \epsilon_\mathrm{p} - \kappa < 1$ for some constant $0 < \kappa < 1$, let $P \geq G = \mathsf{poly}(\lambda)$, $\mathsf{pp} \leftarrow \mathsf{AG.GenParams}(1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n}, G, P)$, and let C be an upper bound on the size of clues generated by $\mathsf{AG.GenClue}$; then $C = \Omega(G)$.

Proof. To prove this, we show that suppose that AG being an AGOMR construction with clue size $o(G)$, we can encode $G = \mathsf{poly}(\lambda)$ bits using $o(G)$ bits. For simplicity, we start with $\epsilon_\mathrm{p} = 2^{-\lambda}, \epsilon_\mathrm{n} = 2^{-\lambda}$.

For $i \in [G]$, let $(\mathsf{sk}_i, \mathsf{pk}_i = (\mathsf{pk}_{\mathsf{clue}\,i}, \mathsf{pk}_{\mathsf{det}\,i})) \leftarrow \mathsf{AG.KeyGen}(\mathsf{pp})$ and a dummy payload p shared by the encoding scheme and the decoding scheme. Given a G-bit string m, let $Y = \{i \mid i \in [G], m_i = 1\}$, $c \leftarrow \mathsf{AG.GenClue}(\mathsf{pp}, (\mathsf{pk}_{\mathsf{clue}\,i})_{i\in Y}, p)$. Let $\mathsf{BB} = (c, p)$. The decoding scheme works as follows:

For $i \in [G]$, compute $\mathsf{PL}_i \leftarrow \mathsf{AG.Decode}(\mathsf{pp}, \mathsf{AG.Retrieve}(\mathsf{pp}, \mathsf{BB}, \mathsf{pk}_{\mathsf{det}\,i}, \bar{k} = 1), \mathsf{sk}_i)$. Let $m'_i = 1$ if $p \in \mathsf{PL}_i$ and 0 otherwise. We then have $m' = m$ except for negligible probability (since both false positive and false negative rates are $2^{-\lambda}$, the error probability is bounded by $G \cdot 2^{-\lambda}$, which is negliglble given $G = \mathsf{poly}(\lambda)$). Notice that c generated by scheme AG of size $o(G)$ is the only information other than payload p published on the BB, we thus recover a G-bit message m with only $o(G)$ information.

Now let $1 > c > 0$ be a constant, and $\epsilon_\mathrm{n} < 1 - \epsilon_\mathrm{p} - c$, we can simply repeat the process above for λ times. To recover each bit, takes the majority (i.e., among the λ repetition, if a bit decodes to 1 for $> \lambda/2$ times, set the final result to be 1 and otherwise 0). Since $\epsilon_\mathrm{n} < 1 - \epsilon_\mathrm{p} - c$, we can recover the message except for negligible probability. $\square$

Lower bound for FGOMR. Similarly, the lower bound holds for FGOMR, formally:

Theorem 4.6. For any FGOMR scheme FG, for any $\lambda > 0$, $0 < \epsilon_\mathrm{n} < 1 - \epsilon_\mathrm{p} - \kappa < 1$ for some constant $0 < \kappa < 1$, let $P \geq G = \mathsf{poly}(\lambda)$, $\mathsf{pp} \leftarrow \mathsf{FG.GenParams}(1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n}, G, P)$, and let C be an upper bound on the size of clues generated by $\mathsf{FG.GenClue}$; then $C = \Omega(G)$.

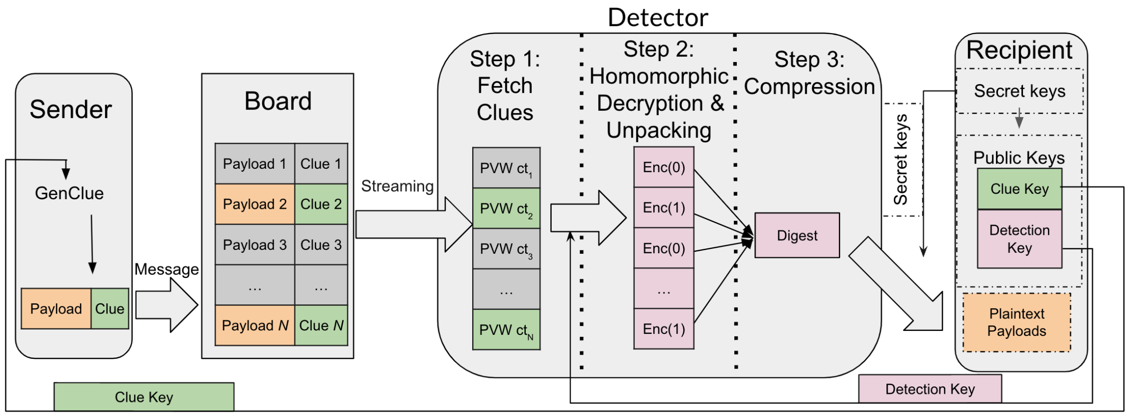


Figure 5: Main components of the original OMR construction. Our main focus in this work is to adapt and optimize the sender side and step 1 on the detector side to the group setting. Step 2 only has very minor changes. Step 3 maintains exactly the same.

Proof. Proving the lower bound for FGOMR is almost the same: except that now to encode a message, the encoder first needs to generate a group clue key using FG.GroupKeyGenAux and FG.GroupKeyGen, but note that since all the sk's are shared by the encoder and decoder, this is easily doable. All the other arguments remain the same as above. □

5 Optimization to Prior Work

This section introduces two optimizations to the original OMR constructions[LT22]. Subsequent sections are based on this optimized OMR construction. We first recall the OMRp2 of [LT22], with its high-level algorithms sketched in Fig. 5 and summarized as follows.

Note that these optimizations are independent of our GOMR constructions. One can understand our GOMR constructions without reading the optimizations in this section, but simply being familiar with the original OMR construction (or read Section 5.1).

5.1 The Original OMR Construction

Setup and sending (OMR.GenClue). A bulletin board BB of size N, $\mathsf{BB}[i] = (x_i, c_i)$, contains messages, each consisting of a payload x_i and a clue c_i. Each clue c_i is generated by the sender of that message by producing a PVW ciphertext of ℓ 1's encrypted under the PVW public key found in the clue key of the intended (secret) recipient.

Retrieval (OMR.Retrieve). To request retrieval, recipient j sends its detection key $\mathsf{pk}_{\mathsf{det}_j} = (\mathsf{BFV.pk}_j, \mathsf{BFV.Enc}(\mathsf{BFV.pk}_j, \mathsf{PVW.sk}_j))$ to the detector. The detector then uses $\mathsf{pk}_{\mathsf{det}_j}$ to homomorphically process all messages and sends back a digest containing payloads addressed to recipient j, via the following stages.

<u>Fetch clues:</u> The first step of the original construction is simply fetching all the clues (and payloads) from the board.

<u>Homomorphic decryption and unpacking:</u> Then, the detector uses the clues on the board to obliviously obtain N encrypted bits, each indicating whether a message is pertinent (0 for no and 1 for yes) (i.e., the left half of Fig. 5), as follows. For each clue c_i, the detector uses the $\mathsf{PVW.sk}_j$ encrypted under $\mathsf{BFV.pk}_j$ to homomorphically decrypt c_i, and then homomorphically computes the pertinency indicator b_i to be 1 if all ℓ decryption results are 1's. Thus, b_i is (with high probability) 1 iff the i-th message is intended for recipient j. For efficiency, this is done in large batches of D clues at a time, using BFV's SIMD evaluation. Let PV denote the resulting vector of BFV-encrypted pertinency bits $(b_1, \ldots, b_D)$. The detector then decomposes PV

through function $\mathsf{PVUnpack}$ into D BFV ciphertexts $\overrightarrow{\mathsf{PV}}_{i\in[D]}$, where $\overrightarrow{\mathsf{PV}}_i = \mathsf{BFV.Enc}(b_i,\ldots,b_i)$ (i.e., a vector of encrypted pertinency bits).

Compression: The detector compresses the encrypted binary vector of length N, which is sparse (mostly 0), to form a digest of size $o(N)$, using coding techniques as follows.

Assuming at most $\bar{k}$ messages are pertinent, the detector randomly assigns each message to one of $m > \bar{k}$ buckets, each consisting of an accumulator and a counter. Suppose messages $S \subseteq [N]$ are assigned to a bucket. The detector then homomorphically sums up the indices of all the assigned pertinent messages in the same bucket as the value of the accumulator (i.e., $\mathsf{Acc} \leftarrow \sum_{j\in X} \overrightarrow{\mathsf{PV}}_j \cdot j$), and records the number of the pertinent messages in each bucket by using the counter (i.e., $\mathsf{Ctr} \leftarrow \sum_{j\in X} \overrightarrow{\mathsf{PV}}_j \cdot 1$) (e.g., result = 2 if and only if there are two pertinent messages assigned to the bucket). The compression succeeds only if there is no more than one pertinent message assigned in one bucket (i.e., the counter value is 0 or 1). Otherwise, it is a collision. To amplify the success probability, the detector will repeat this process several times. We denote the number of such repetitions by C. [LT22, §6.1.2] further uses partial information gathering (i.e., gather the information from the buckets whose $\mathsf{Ctr} = 1$ in each repeaitition) to reduce C.

The payloads can also be compressed in the same way as the pertinent indices described above. However, for better efficiency, [LT22, §6.3] uses Sparse Random Linear Coding (SRLC). See details of the payload processing there (we reuse this component verbatim).

Decryption (OMR.Decode). The recipient, given the digest, decrypts and decodes it and obtains the correct payloads using the PVW secret key $\mathsf{PVW.sk}$.

This work focuses on optimizing the detector's cost in the aforementioned stage "Homomorphic decryption and unpacking", which dominates cost — in the case where a message is addressed to a group of recipients. To this end, we also modify the sender's algorithm in the first stage. The later stages are essentially the same as [LT22], except for the optimization introduced in Section 5.3.

5.2 Optimizing PVUnpack

The construction in [LT22, §7.4] expands a packed PV (i.e., a BFV ciphertext encrypting $\{0,1\}^D$) into D BFV ciphertexts slot by slot (i.e., extract a single slot into a BFV ciphertext and replicate the result into all the D slots), resulting in $O(D\log(D))$ homomorphic operations. We improve this by batching the expansion step, resulting in $O(D)$ homomorphic operations.

The replication of the bits across all slots takes $\log(D)$ homomorphic rotations per slot, resulting in $D\log(D)$ rotations and D multiplications (see [LT22, Alg. 6]). We observe that these rotations can be amortized over multiple slots:

Instead of extracting one slot and rotate $\log(D)$ times, we instead extract half of the slots (thus resulting into 2 ciphertexts) and rotate once, and then extract 1/4 of the slots (thus resulting into 4 ciphertexts) and then rotate them once, and repeat this process totally $\log(D)$ levels. In the end, we obtain D ciphertexts each encrypting a slot as expected. This is essentially a binary tree structure, with $\log(D)$ depth.

However, this also means that the multiplicative depth is $\log(D)$. Thus, instead of having a binary tree, we can have a t-ary tree (i.e., a tree with a branching factor of t instead of 2), resulting in a depth of $\log_t(D)$. More generally, we do not need the same branching factor at each level. Instead, we have branching factors $(b_1,\ldots,b_L) \in \mathbb{Z}_D^L$, where $b_1 \times \cdots \times b_L = D$ for the tree with L levels. For simplicity, we only have branching factors that are power-of-two. This general tree structure gives us in total $D\log(x_L) + \frac{D}{x_{L-1}}\log(\frac{x_{L-1}}{x_L}) + \cdots + \frac{D}{x_1}\log(\frac{x_1}{x_2})$ rotations, where $x_i = \prod_{j=1}^{i} b_i$.

We formalize the procedure in Algorithm 1. [4]

[4] For simplicity and concrete performance, when invoking our new $\mathsf{PVUnpack}$ in our OMR/GOMR constructions, we always invoke it with $L = 2$. Thus, the interface become the same as the old $\mathsf{PVUnpack}$.

Algorithm 1 Optimized $\mathsf{PVUnpack}$

procedure $\mathsf{PVUnpack}(\mathsf{pp}_{\mathsf{BFV}}, \mathsf{pk}_{\mathsf{BFV}}, \mathsf{ct}, L)$
  Initialize $\mathsf{res} \leftarrow (\mathsf{ct})$ as a vector
  Find power-of-two points $(x_1, \ldots, x_L)$ s.t. $D\log(x_L) + \frac{D}{x_{L-1}}\log(\frac{x_{L-1}}{x_L}) + \cdots + \frac{D}{x_1}\log(\frac{x_1}{x_2})$ is minimal
  for $i \in x$ **do**
    $\mathsf{tmp} = (1, \ldots, 0)$
    $\triangleright$ where $(1 + k \cdot x_{i-1}, \ldots, x_i + k \cdot x_{i-1})$ are 1's for all $k \in [D/x_{i-1}]$, for $i = 1$, $k \in \{0\}$, $x_0 = 0$
    $\mathsf{cur\_size} \leftarrow \frac{D}{x_{i-1}}$ $\triangleright$ $x_0 = D$
    Initialize res' as an empty vector
    for $u = 1$ to $\mathsf{cur\_size}$ **do**
      $\mathsf{res}_{\mathsf{tmp}} \leftarrow \mathsf{res}[u]$
      for $j = 0$ to $\frac{x_{i-1}}{x_i} - 1$ **do**
        $\mathsf{tmp} = (1, \ldots, 0)$ $\triangleright$ where $(1 + k \cdot x_{i-1} + j \cdot \frac{x_{i-1}}{x_i}, \ldots, x_i + k \cdot x_{i-1} + j \cdot \frac{x_{i-1}}{x_i})$ are 1's for all $k \in [D/x_{i-1}]$
        $\triangleright$ for $i = 1$, $k \in \{0\}$, $x_0 = D$
        $\mathsf{res}'_{\mathsf{tmp}} \leftarrow \mathsf{BFV.Eval}(\mathsf{pk}_{\mathsf{BFV}}, \mathsf{res}_{\mathsf{tmp}}, \mathsf{tmp}, \times)$
        for $\beta = \log(x_i)$ to $\log(x_{i-1})$ **do** $\triangleright$ $x_0 = D$
          $\mathsf{res}'_{\mathsf{tmp}} \leftarrow \mathsf{BFV.Rotate}(\mathsf{pk}_{\mathsf{BFV}}, \mathsf{res}'_{\mathsf{tmp}}, 2^\beta)$
        Append $\mathsf{res}'_{\mathsf{tmp}}$ to res'
    $\mathsf{res} \leftarrow \mathsf{res}'$
  return res

5.3 Optimizing OMRp2

Another optimization is tightening the parameter choice of m (the aforementioned number of buckets for index retrieval) and C (the aforementioned number of repetitions to reduce failure probability) of $\mathsf{OMRp2}$ in [LT22, §6.1.2]. After our optimization, $\mathsf{OMRp2}$ dominates $\mathsf{OMRp1}$ for most practical parameters (including the parameters set by [LT22, §10]) unless $\bar{k}$ is huge and N is relatively small.

Originally, $\mathsf{OMRp2}$ chooses a fixed $m = D$ when $\bar{k} \ll D$,[5] and then chooses the smallest C such that the error probability is achieved (i.e., $1 - \prod_{i=1}^{\bar{k}-1}(1 - (\frac{i}{m})^C) \leq \epsilon_{\mathrm{p}}/4$). Instead, we let m vary and minimize $m \cdot C$ subject to the same constraint. Since each bucket takes $\log_t(N) + 1$ BFV slots to store the accumulator and the counter, this reduces the number of BFV ciphertexts from $(\log_t(N) + 1) \cdot C \cdot m/D = (\log_t(N) + 1) \cdot C$ to $(\log_t(N) + 1) \cdot m \cdot C/D$ ciphertexts; or concretely: from 15 to 1 for the parameters in [LT22, §10].

Therefore, the optimized $\mathsf{OMRp2}$ performance will dominate $\mathsf{OMRp1}$ for most parameters.

This optimized $\mathsf{OMRp2}$, dubbed $\mathsf{OMR\text{-}OPT}$, is given in Algorithm 2.

Theorem 5.1. The scheme $\mathsf{OMR\text{-}OPT}$ in Algorithm 2 is an OMR scheme for $N < D \cdot t/2$, assuming security of LWE encryption and security of BFV leveled HE as in [LT22], when instantiated with PRF f and an SRLC scheme SRLC (Definition B.1). Moreover when instantiated with $\mathsf{SRLC1}$ ([LT22, Alg 3]), $\mathsf{OMR\text{-}OPT}$ is also compact.

Proof sketch. The only difference between $\mathsf{OMRp2}$ defined in [LT22, §7.4] and $\mathsf{OMR\text{-}OPT}$ is the optimized accumulator and counter encoding described in Section 5.3.

The proof of soundness, privacy, and compactness remain the same as in [LT22, Thm 7.2]. Thus, we only need to show that the probability of the optimized accumulator/counter overflowing is $\leq \epsilon_{\mathrm{n}}/4$, which will then satisfy the completeness requirement defined in [LT22, §4.3].

From the above analysis, we have $d = \lfloor \frac{\bar{d}D}{\lceil \log_t(N) \rceil + 1} \rfloor$ buckets, each expected to be assigned at most N/d messages, as the number of pertinent messages detected is trivially bounded by N. But bucket counters

[5] When $\bar{k} \not\ll D$, simply concatenate c multiple BFV ciphertexts together and view them as a single ciphertext, thus having $D \leftarrow c \cdot D$.

may overflow, i.e., get incremented by more than t assigned messages detected as pertinent (whether true positives or false positives). We bound this overflow probability as follows.

$$
\begin{aligned}
\Pr[X \geq t] &< \Pr[X \geq 2N/D] \quad (\text{since } N < Dt/2)) \\
&= \Pr[X \geq 2(N/D)] \\
&= \Pr\left[X \geq 2\left(\frac{N}{d \cdot (\lceil \log(N) \rceil + 1)}\right)\left(\frac{d \cdot (\lceil \log(N) \rceil + 1)}{D}\right)\right] \\
&\leq \exp\left(-\frac{\delta^2}{2+\delta}\frac{N}{d(\lceil \log(N) \rceil + 1)}\right) \quad (\text{by Chernoff bound, where } \delta = \frac{2d(\lceil \log(N) \rceil + 1)}{D} - 1 = 2\bar{d} - 1) \\
&= \exp\left(-\frac{\delta^2}{2+\delta}\frac{N}{\bar{d}D}\right) \\
&\leq \exp(-\frac{(2\bar{d}-1)^2}{2\bar{d}+1}\frac{t/2}{\bar{d}})
\end{aligned}
$$

By the union bound, the probability of none of the d buckets overflowing is $d \exp(-\frac{(2\bar{d}-1)^2}{2\bar{d}+1}\frac{t/2}{\bar{d}}) < \epsilon_{\mathrm{n}}/4$, where $d = O(\log(\epsilon_{\mathrm{n}}^{-1}))$. Therefore, for $N < Dt/2$, the condition at line 23 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 $k \leq \bar{k}$. $\square$

Due to these optimizations, our improved OMR-OPT is more efficient than OMRp2 in [LT22] for most applications. Thus, our GOMR construction will be based on OMR-OPT.

6 Ad-hoc Group OMR

We proceed to construct Ad-hoc Group OMR (AGOMR), where a sender can send each message to any arbitrary set of up to G recipients, as defined in Section 4.1.

6.1 Trivial Solution

AGOMR can be directly but inefficiently realized using any OMR scheme (as in [LT22]): for each group message, and each of its G recipients, the sender sends a separate single-recipient OMR message with a clue dedicated to that recipient (i.e., a PVW ciphertext in the case of [LT22, §7]). The number of clues in the board then increases by a factor of G, and therefore so does the detector's work.

When the above construction is instantiated with OMRp2 in [LT22, §7.4] or OMR-OPT of Section 5, the detector first performs G homomorphic PVW decryption to obtain G encrypted bits per message. If the message is pertinent, one of these encrypted bits would be 1, otherwise the bits would be all 0's (with high probability).

After the bits are obtained, the detector performs the remaining procedure G times per message. This is wasteful, since there is at most one 1 in G bits. Ideally, the detector can sum up all G bits and performs the later procedures once per message. However, with a non-negligible false positive rate and the probability of malicious clues, there are potentially multiple 1's, which would cause retrieval failure. This can be solved by additional changes, but the cost remains dominated by the aforementioned blowup in the number of homomorphic PVW decryptions. [6]

[6] This scheme is benchmarked under the label "OMR" in Fig. 10.

Algorithm 2 $\mathsf{OMR\text{-}OPT}$: Improved Practical Compact Oblivious Message Retrieval

procedure $\mathsf{OMR\text{-}OPT.GenParams}(1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n})$
  Choose $\mathsf{pp}_\mathsf{BFV} = (D, t, \dots)$, $\mathsf{pp}_\mathsf{PVW} = (n, w, \ell, q, \sigma)$, and range r with one change:
  Replace item 3 with $\ell \cdot (1 - \mathsf{erf}(r/(\sqrt{2w}\sigma))) < \epsilon_\mathrm{n}/4$ ▷ See *Setting parameters*
  return $\mathsf{pp} = (1^\lambda, \epsilon_\mathrm{n}, \epsilon_\mathrm{p}, \mathsf{pp}_\mathsf{BFV}, \mathsf{pp}_\mathsf{PVW}, r)$ ▷ Provided implicitly below
procedure $\mathsf{OMR\text{-}OPT.KeyGen}$
  $(\mathsf{sk}_\mathsf{pvw}, \mathsf{pk}_\mathsf{pvw}) \leftarrow \mathsf{LWE.KeyGen}()$
  $(\mathsf{sk}_\mathsf{BFV}, \mathsf{pk}_\mathsf{BFV}) \leftarrow \mathsf{BFV.KeyGen}()$
  $\mathsf{ct}_\mathsf{pvwSK} \leftarrow \mathsf{BFV.Enc}(\mathsf{pk}_\mathsf{BFV}, \mathsf{sk}_\mathsf{pvw})$
  return $(\mathsf{sk} = (\mathsf{sk}_\mathsf{BFV}), \mathsf{pk} = (\mathsf{pk}_\mathsf{clue} = \mathsf{pk}_\mathsf{pvw}, \mathsf{pk}_\mathsf{det} = (\mathsf{pk}_\mathsf{BFV}, \mathsf{ct}_\mathsf{pvwSK})))$
procedure $\mathsf{OMR\text{-}OPT.GenClue}(\mathsf{pk}_\mathsf{clue}, x)$
  $\vec{m} \leftarrow (0, \dots, 0) \in \mathbb{Z}_t^\ell$
  $c \leftarrow \mathsf{LWE.Enc}(\mathsf{pk}_\mathsf{clue}, \vec{m})$ ▷ Recall: clue $c \in \mathbb{Z}_t^{n \times \ell}$
  return c
procedure $\mathsf{OMR\text{-}OPT.Retrieve}(\mathsf{BB}, \mathsf{pk}_\mathsf{det}, \bar{k})$
  ▷ **Phase 1: Initialization**
  Draw a random seed $s = (s_f, s_h)$
  Parse $\mathsf{BB} = \{(x_1, c_1), \dots, (x_N, c_N)\}$ and $\mathsf{pk}_\mathsf{det} = (\mathsf{pk}_\mathsf{BFV}, \mathsf{ct}_\mathsf{pvwSK})$
  Let $C \leftarrow N/(D \cdot \log(t))$
  Initialize $(\mathsf{Acc}_i = \mathsf{BFV.Enc}(\mathsf{pk}_\mathsf{BFV}, (0, \dots, 0)))_{i \in [C]}$ ▷ D zeros
  ▷ **Phase 2: detection** in batches of D messages
  Find the smallest $d', \bar{d}$ and let $d = \lceil \frac{\bar{d}D}{\lceil \log(N) \rceil + 1} \rceil$ such that:
  (1) $1 - \prod_{i=1}^{\bar{k}-1}(1 - (\frac{i}{d})^{d'd}) < \epsilon_\mathrm{n}/4$
  (2) $d \cdot \exp(-\frac{N(2\bar{d}-1)^2}{(2\bar{d}+1)\bar{d}D}) \le \epsilon_\mathrm{n}/4$
  Initialize $(C_{\mathsf{lhs},z} \leftarrow (\mathsf{BFV.Enc}(\mathsf{pk}_\mathsf{BFV}, (0, \dots, 0))_w)_{w \in [d']}$
    ▷ encrypted $d'\bar{d}D$ zeros as a single concatenated BFV ciphertext, to avoid detailed mod calculation
  for $i = 1$ to N/D **do** ▷ Assume wlog that D divides N
    Parse each clue as $c_{iD+i'} = ((c_{i',\kappa}))_{\kappa \in [n+\ell]} \in \mathbb{Z}_q^{n+\ell}$ for $i' \in [D]$
    $\bar{c}_\kappa \leftarrow (c_{i',\kappa})_{i' \in [D]}$ for $\kappa \in [n + \ell]$
    ▷ $\bar{c}_\kappa$ lists the κ-th element of every PVW clue in this batch

    $\alpha_1 \leftarrow \mathsf{InnerProd}(\mathsf{pp}_\mathsf{BFV}, \mathsf{pk}_\mathsf{BFV}, \mathsf{ct}_\mathsf{pvwSK}, (\bar{c}_\kappa)_{\kappa \in [n+\ell]})$
    $\alpha_2 \leftarrow \mathsf{RangeCheck}(\mathsf{pp}_\mathsf{BFV}, \mathsf{pk}_\mathsf{BFV}, \alpha_1, r)$
    $\alpha_3 = \prod_{i=0}^{\ell-1}(1 - \alpha_2[i])$
    $(\mathsf{PV}_{i'})_{i' \in ((i-1)D, iD]} \leftarrow \mathsf{PVUnpack}(\mathsf{pp}_\mathsf{BFV}, \mathsf{pk}_\mathsf{BFV}, \alpha_3)$
  for $i = 1$ to N **do**
    $j, k \leftarrow f_{s_f}(i)$ ▷ $j \in [C]$, $k \in [d]$
    $a \leftarrow (k-1) \cdot \lceil \log(N) + 1 \rceil$
    $b \leftarrow k \cdot \lceil \log(N) + 1 \rceil$
▷ We need to update slot $[a, b-1]$; first $b - a - 2$ slots are for accumulator and last one is for counter
    $C_{\mathsf{lhs},j}[a, b-2] = C_{\mathsf{lhs},j}[a, b-2] + i \cdot \mathsf{PV}_i$ ▷ t-ary addition
    $C_{\mathsf{lhs},j}[b-1] = C_{\mathsf{lhs},j}[b-1] + 1 \cdot \mathsf{PV}_i$ ▷ Slot-wise addition, done homomorphically
  ▷ **Phase 3: Finalization**
  $\hat{k} \leftarrow \bar{k} + N \log(N) \epsilon_\mathrm{p}$
  $(\mathsf{pp}_\mathsf{SRLC}, m) \leftarrow \mathsf{SRLC.GenParams}(1^\lambda, \hat{k}, \epsilon_\mathrm{n}/4, t)$
    ▷ In practice $(\mathsf{pp}_\mathsf{SRLC}, m)$ is preprocessed and tabulated and therefore becomes $O(1)$
  Initialize combinations $\{\mathsf{Cmb} = \mathsf{BFV.Enc}(\mathsf{pk}, 0)\}_{k \in [m]}$
  for $i = 1$ to N **do**
    $S \leftarrow \mathsf{SRLC.GenWeights}(\mathsf{pp}_\mathsf{SRLC})$
    for $(j, w_j) \in S$ **do**
      $\mathsf{Cmb}_j = \mathsf{Cmb}_j + \mathsf{PV}_i \cdot x_i \cdot w_j$ ▷ by homomorphic addition and scalar multiplication
  return $M = (s, (\mathsf{Acc}_{\mathsf{lhs},i})_{i \in [C \cdot d/D]}, (\mathsf{Ctr}_{\mathsf{lhs},i})_{i \in [C \cdot d/D]}, (\mathsf{Cmb}_k)_{k \in [m]}, \mathsf{pp}_\mathsf{SRLC})$
procedure $\mathsf{OMR\text{-}OPT.Decode}(\mathsf{pp}, M, \mathsf{sk})$
  As in [LT22], details omitted.

6.2 Improved Ad-hoc Group OMR

The main detection cost, in the trivial scheme above, lies in the G-fold increase in expensive homomorphic operations: homomorphic decryption of PVW ciphertexts and the digest compression process. In this section, we use a different technique to construct AGOMR scheme so that the decryption process and digest compression only need to be performed once for each message. (This is generalized to the case of multiple payloads in Section 6.3.)

Polynomial interpolation encoding. Let $\mathcal{D}$ be a finite field (to be fixed later). Each recipient randomly draws $\mathsf{id} \leftarrow_\$ \mathcal{D}$ as its unique public identity and includes the id as part of its $\mathsf{pk}_{\mathsf{clue}}$. To send a payload to a group of G recipients, the sender first generates G PVW ciphertexts: $(c_i \in \mathbb{Z}_t^{n+\ell})_{i\in[G]}$ all encrypting 1's using the corresponding PVW public keys included in the $\mathsf{pk}_{\mathsf{clue}}$ of those G recipients. But instead of directly sending these G PVW ciphertexts as clues (as in the trivial solution), the sender encodes them into a polynomial to be used as the clue. Specifically, the sender interpolates a degree-$(G-1)$ polynomial $f : \mathcal{D} \to \mathbb{Z}_t^{n+\ell}$ such that $f(\mathsf{id}_i) = c_i$ for $i \in [G]$, and publishes the coefficients of this polynomial as the clue. To guarantee all the id's are unique, $\mathcal{D}$ needs to be large enough (e.g., $\mathcal{D} = \mathbb{Z}_{2^{127}-1}$).

During retrieval, the detector takes the recipient's id included in the detection key. For each message $(x_i, f_i)_{i\in[N]}$, it evaluates $f_i(\mathsf{id})$ (in plaintext form), to obtain a PVW ciphertext. Thus, if id is one of the points over which f is interpolated, then $f(\mathsf{id})$ is a PVW ciphertext encrypting 1^ℓ. Otherwise, $f(\mathsf{id})$ is an independently and pseudorandomly sampled vector of size $\mathbb{Z}_t^{n+\ell}$ w.r.t to this recipient's decryption key and will be decrypted to 1^ℓ with probability $\leq \epsilon_{\mathrm{p}}$. Hence, after obtaining the PVW ciphertext (i.e., $f(\mathsf{id})$), we proceed as in the detection procedure of $\mathsf{OMR\text{-}OPT}$, performing a *single* homomorphic decryption of a PVW ciphertext .

Since the PVW ciphertexts are computationally indistinguishable from uniformly-drawn vectors in $\mathbb{Z}_t^{n+\ell}$, the interpolated polynomial is thus indistinguishable from a random polynomial, in the absence of recipients' secret keys. Even to an intended recipient, other intended recipients' identities are unknown. Computational privacy follows.

Construction 6.1. The above yields an AGOMR scheme, $\mathsf{AGOMR1}$ (instantiated with $\mathsf{OMR\text{-}OPT}$), which achieves computational privacy and in whose detection is particularly efficient — since the extra computation to handle groups is a mere polynomial evaluation in plaintext. However, in this construction each retrieval query reveals the querying recipient (though not their pertinent messages) by including the identity id in the detection key, so it does not achieve detection-key unlinkability (Definition 4.3). The following constructions address this and achieve key-unlinkability.

Detection-key unlinkability for the poly-interpolation-based method. To acheive detection-key unlinkability, the biggest challenge is that now the detection key includes an id, other than the BFV public key and the encrypted PVW secret key. Thus, instead of evaluating $f(\mathsf{id})$ in plaintext, the recipient also encrypts id using BFV public key and evaluate $f(\mathsf{id})$ homomorphically.

Thus, $\mathsf{RegenDetectKey}$ (in Definition 4.3) is to freshly generate new BFV public keys, encrypting the PVW secret key and the id.

However, in this case, the function f is also evaluated under FHE, so it needs to be FHE-friendly.

FHE-friendly linear encoding. As our schemes are based on BFV leveled-HE scheme, the multiplicative depth needs to be small (e.g., ≤ 30 levels) and the plaintext modulus to be reasonable (normally, 16–50 bits, and preferably ≤ 20 bits given our detector circuit). However, homomorphically evaluating the polynomial above requires a multiplicative depth to be $\log(G)$, and requires the FHE plaintext space to be a large finite field (e.g., $\mathbb{Z}_{2^{127}-1}$). While the multiplicative depth can be reduced to 1 by sending $\mathsf{Enc}(\mathsf{id}), \mathsf{Enc}(\mathsf{id}^2), \ldots, \mathsf{Enc}(\mathsf{id}^G)$, the large plaintext modulus remains to be an issue. Therefore, we change the identities to be vectors $\mathsf{id} \in \mathbb{Z}_t^I$ for some much smaller t (with some large enough I to be fixed later), and then interpolate a *linear* function $f : \mathbb{Z}_t^I \to \mathbb{Z}_t^{n+\ell}$ instead. In this case, the function f can be represented by a matrix $\mathsf{M} \in \mathbb{Z}_t^{I\times(n+\ell)}$. To recover the clue, the detector just needs to compute clue $c_i = \mathsf{id}_i \times \mathsf{M}$, which is a homomorphic matrix multiplication of multiplicative depth 1.

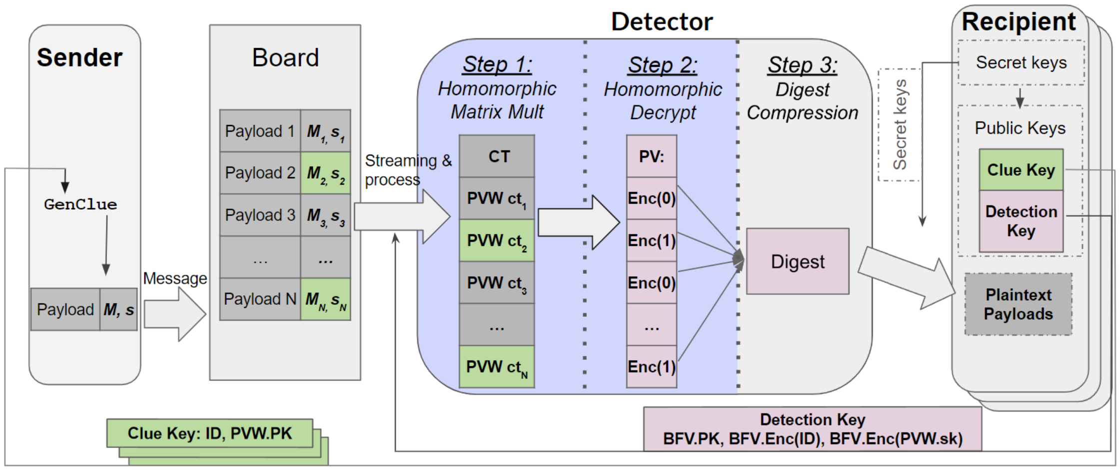


Figure 6: Main components of our Ad-hoc Group OMR construction.

Construction 6.2. The above yields an AGOMR scheme (instantiated with OMR-OPT), AGOMR2, which is a fully secure, efficient, but weaker, in the following sense. By encrypting the id vectors, this scheme satisfies the detection-key unlinkability. However, completeness of this scheme requires the groups to be honestly formed; specifically, the groups are formed independent of the participant id values (details below). Whether assumption is apt depends on the underlying applications, and at worst, a group that cannot form a clue due to maliciously chosen ids can be separated into multiple subgroups to avoid the issue. A better solution is discussed next.

Dealing with a maliciously formed group of ID. For such a matrix M as above to exist, we need all the G id's used to generate the clue to be linearly independent. Since all id's are randomly generated, for an honestly formed group, the probability that those randomly chosen G id's are linearly independent is roughly $1 - t^{(G-I)}$ for $I > G$ by [SGGC14b, Lemma 1]. However, the groups may be formed maliciously after seeing the id's. To mitigate this, we increase the length I of the id's. Let ϵ_{DI} denote the probability that given P randomly drawn id's, there exists a combination of G of those id's are linearly dependent. By union bound, $\epsilon_{\mathrm{DI}} = \binom{P}{G} \prod_{I-G+1}^{I} (1 - 1/t^i)$. Setting $I = O(G \log(P))$ suffices to achieve $\epsilon_{\mathrm{DI}} \leq \mathsf{negl}(\lambda)$. To send to G recipients, the sender then solves for $M \in \mathbb{Z}_t^{I \times (n+\ell)}$ as before.

Compressing the clue. With the enlarged I, publishing the whole matrix M as clue results in a $O(G \log(P))$ clue size, which is costly (concretely and asymptotically) when the number P of total recipients is large. To reduce its size, we use pseudorandomness derived from a succinct seed, as follows. The sender draws a random seed s, and uses it to generate a pseudorandom matrix $Z \in \mathbb{Z}_t^{I \times G'}$ (where $G' \geq G$ is set below). The sender computes $C \leftarrow W \times Z \in \mathbb{Z}_t^{G \times G'}$, which is indistinguishable from uniform random as shown in Theorem 6.3, where $W = \begin{pmatrix} \mathsf{id}1 \\ \vdots \\ \mathsf{id}_G \end{pmatrix}$. Let ϵ_{DS} denote the probability that C is not full rank given that W is full rank. Since C is computationally indistinguishable from random, again, by [SGGC14b, Lemma 1], $\epsilon_{\mathrm{DS}} = \prod_{G'-G+1}^{G'} (1 - 1/t^i)$. Thus, to make $\epsilon_{\mathrm{DS}} = \mathsf{negl}(\lambda)$, we have $G' = G + O(\lambda)$. Then, the sender computes the matrix $\mathsf{M_s} \in \mathbb{Z}_t^{G' \times (n+\ell)}$ such that $C \times \mathsf{M_s} = (c_1, \ldots, c_G)^\intercal$. The clue is of the form $(\mathsf{M_s}, \mathsf{s})$, which is of size $O(G)$.

Lemma 6.3. For any $0 < G \leq \bar{G}$, given any matrix $A \in \mathbb{Z}_t^{G \times \bar{G}}$ with rank G, the distribution $D = \{Y : Y = A \times Z, Z \leftarrow_\$ \mathbb{Z}_t^{\bar{G} \times G'}\}$ is equivalent to the uniform distribution $D_U = \{Y : Y \leftarrow_\$ \mathbb{Z}_t^{G \times G'}\}$.

Proof sketch. As A has full row rank, $\mathsf{dim}(\mathrm{kernel}(A)) = t^{\bar{G} \times G'} / t^{G \times G'} = t^{G'(\bar{G}-G)}$. Therefore, for any $Y \in$

$\mathbb{Z}^{G \times G'}$, $|\{Z \mid Y = A \times Z\}| = \mathsf{dim}(\text{kernel}(A))$. Therefore, if we sample an $Z \in \mathbb{Z}_t^{\bar{G} \times G'}$ uniformly at random, for any $Y \in \mathbb{Z}^{G \times G'}$, $\Pr[A \times Z = Y] = \frac{1}{t^{G \times G'}}$. $\square$

Detector operations. The detector uses s to recover Z and homomorphically computes $(\mathsf{id} \times Z) \times \mathsf{M}_\mathsf{s}$ to recover the PVW ciphertext c, where id is encrypted under the recipient's BFV public key included in pk_det. It then homomorphically decrypts the PVW ciphertext and processes digest encodings, as in OMR-OPT.

Efficiency analysis. With the technique against maliciously chosen groups, the computation cost of recovering the clue grows to $\tilde{O}(G \cdot (G \log(P) + n + \ell))$. However, compared to the baseline scheme OMRp2, which requires G expensive PVW ciphertext decryption, involving expensive range checks, and another G expensive digest compression, those computations are all relatively cheap plaintext-by-ciphertext multiplications. The resulting scheme is thus much lighter w.r.t. to the computation work. Note that compared to the trivial solution, our construction does not reduce the clue size. Later in Section 7, we show that the construction under the fixed-group setting can also have the merit of a smaller clue size. See Fig. 10a for concrete comparisons in terms of the clue sizes.

Combining all of the above, we obtain our main AGOMR construction AGOMR3 given in Algorithm 3. Fig. 6 portrays the high-level components of the resulting scheme. Compared to Fig. 5, the major changes are the clues and step 1. Steps 2 and 3 are unchanged.

Theorem 6.4. The scheme AGOMR3 in Algorithm 3 is an AGOMR scheme for $N < D \cdot t/2$, $P = \mathsf{poly}(\lambda)$, and $G = \mathsf{poly}(\lambda)$, assuming hardness of LWE and security of BFV leveled HE; when instantiated with PRF f and an SRLC scheme SRLC (Definition B.1). Moreover when instantiated with SRLC1 ([LT22, Alg 3]), AGOMR3 is also compact. Furthermore, AGOMR3 is detection-key-unlinkable.

Above, *SRLC (Sparse Random Linear Code)* refers to schemes for encoding values as linear combinations drawn from certain distributions, as defined in [LT22, §6.3]. SRLC1 refers to a specific such scheme, defined in [LT22, Alg 3]. These are internal components inherited from [LT22], used in parts of *compression* phase of OMR that we inherit unaltered. We include them in Appendix B for reference.

Proof sketch. Completeness: Most part of the completeness is the same as the completeness proof of Theorem 5.1 . The only additional argument needed is that we now need the clue for the pertinent message can be successfully generated given any subset of recipients, by solving the matrix M with compressed id matrix $\tilde{\mathsf{id}}$ defined at Algorithm 3 line 22.

Based on Theorem 6.3 and the parameter choice at line 4, the matrix $\tilde{\mathsf{id}}$ is of full rank with probability $(1 - \mathsf{negl}(\lambda)) \cdot \prod_{i=G'-G-1}^{G'} (1 - 1/t^i)$ [SGGC14a, Lemma 1]. Thus, by line 5, $\tilde{\mathsf{id}}$ is full rank with probability $1 - \mathsf{negl}$. Therefore, AGOMR3.GenClue can successfully compute M via a solvable linear equation system with $1 - \mathsf{negl}$ probability.

The remainder of the completeness argument is as in [LT22, Thm 7.2].

Soundness: For some $j \in [N]$, for $1 \notin X_j$,[7] given the random id_1 independently generated from M_i, $\mathsf{id}_1 \times \mathsf{M}_i$ results in a vector $(\vec{a}, \vec{b})$ that is indistinguishable from a vector sampled uniformly at random $\in \mathbb{Z}_t^{n+\ell}$. In this case, false positives occur when the inner products of all ℓ parts of this PVW ciphertext with $\mathsf{sk}_{\mathsf{PVW}_1}$ fall into range $\pm r$. This has probability $((2r + 1)/t)^\ell \le \epsilon_\text{p}$.

Computational privacy: Consider the following two hybrids (differences highlighted in **bold**). We assume w.l.o.g. that $|Y_0| = |Y_1| = G$ (as otherwise, GenClue pads the keys with dummy recipients).

H_0 (the real construction): The challenger, after seeing Y_0, Y_1 and $Z = Y_0 \cap Y_1$ sent from the adversary, sends $(\mathsf{sk}_i)_{i \in Z}$ to the adversary, and receives $Z' \subseteq Z, (\mathsf{sk}_i, \mathsf{pk}_i)_{i \in Z'}$ back. Then, the challenger draws $b \leftarrow \{0, 1\}$ to select Y_b. Then, it generates PVW ciphertexts $\mathsf{ct}_i \in \mathbb{Z}_t^{n+\ell}$ using pk_i for all $\boldsymbol{i \in Y_b}$ (where the keys in Z' are overwritten by the adversary). Form the matrix $\tilde{\mathsf{id}}$ with the id_i of each recipients $i \in Y_b$ as on line 22, and the challenger computes the matrix M using $\tilde{\mathsf{id}}$ and $(\mathsf{ct}_i)_{i \in [G]}$ as on line 23. Finally, the challenger sends back M and the seed s to compress the ids.

[7] Recall that in Definition 4.2, the board generation process output the corresponding ground truth for recipient 1 (without loss of generality), and thus our soundness argument is with respect to recipient 1 as well.

Algorithm 3 $\mathsf{AGOMR3}$: Compact Ad-hoc Group Oblivious Message Retrieval

1: **procedure** $\mathsf{AGOMR3.GenParams}(1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n}, G, P)$
2: Generate $\mathsf{pp}_\mathsf{BFV}, \mathsf{pp}_\mathsf{PVW}$ as in $\mathsf{OMR\text{-}OPT}$
3: If $t \le G^2$, choose $t = G^2$ and pp_PVW accordingly.
4: Choose smallest I s.t. $\binom{P}{G} \cdot \prod_{j=I-G-1}^{I}(1 - 1/t^j) \le 2^{-\lambda}$,
5: and smallest $G' \ge G$ s.t. $\prod_{i=G'-G-1}^{G'}(1 - 1/t^i) \le 2^{-\lambda}$.
6: **return** $\mathsf{pp} = (1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n}, G, \mathsf{pp}_\mathsf{BFV}, \mathsf{pp}_\mathsf{PVW}, G', I)$
7: **procedure** $\mathsf{AGOMR3.KeyGen}(\mathsf{pp})$
8: $\mathsf{id} \leftarrow_\$ \mathbb{Z}_t^I$
9: $(\mathsf{sk}_\mathsf{PVW}, \mathsf{pk}_\mathsf{PVW}) \leftarrow \mathsf{PVW.KeyGen}(\mathsf{pp}_\mathsf{PVW})$
10: $(\mathsf{sk}_\mathsf{BFV}, \mathsf{pk}_\mathsf{BFV}) \leftarrow \mathsf{BFV.KeyGen}(\mathsf{pp}_\mathsf{BFV})$
11: $\mathsf{ct}_\mathsf{SWK} \leftarrow \mathsf{BFV.Enc}(\mathsf{pk}_\mathsf{BFV}, \mathsf{sk}_\mathsf{PVW})$
12: $\mathsf{ct}_\mathsf{ID} \leftarrow \mathsf{BFV.Enc}(\mathsf{pk}_\mathsf{BFV}, \mathsf{id})$
13: **return** $(\mathsf{sk} = (\mathsf{sk}_\mathsf{BFV}, \mathsf{sk}_\mathsf{PVW}, \mathsf{id}), \mathsf{pk} = (\mathsf{pk}_\mathsf{clue} = (\mathsf{id}, \mathsf{pk}_\mathsf{PVW})), \mathsf{pk}_\mathsf{det} = (\mathsf{pk}_\mathsf{BFV}, \mathsf{ct}_\mathsf{ID}, \mathsf{ct}_\mathsf{SWK}))$
14: **procedure** $\mathsf{AGOMR3.GenClue}(\mathsf{pp}, \{\mathsf{pk}_{\mathsf{clue}\,i} = (\mathsf{id}_i, \mathsf{pk}_{\mathsf{PVW}\,i})\}_{i\in[G]}, x)$
15: ▷ To address less than G, pad with dummy parties using $\mathsf{AGOMR3.KeyGen}$
16: **for** $i = 1 \cdots G$ **do**
17: $\vec{m} \leftarrow (0, \ldots, 0) \in \mathbb{Z}_t^\ell$
18: $c_i \leftarrow \mathsf{LWE.Enc}(\mathsf{pp}_\mathsf{PVW}, \mathsf{pk}_{\mathsf{PVW}\,i}, \vec{m})$ ▷ All encryptions are done with distinct randomness
19: ▷ Recall: clue $c_i \in \mathbb{Z}_t^{n\times\ell}$
20: Draw a random seed s
21: Use s to sample Z from $\mathbb{Z}_t^{I\times G'}$
22: $\tilde{\mathsf{id}} = \begin{pmatrix} \mathsf{id}_1 \\ \vdots \\ \mathsf{id}_G \end{pmatrix} \cdot Z$ ▷ $\tilde{\mathsf{id}} \in \mathbb{Z}_t^{G\times G'}$
23: Solve for a matrix $\mathsf{M} \in \mathbb{Z}_t^{G'\times(n+\ell)}$ s.t. $\tilde{\mathsf{id}} \cdot \mathsf{M} = \begin{pmatrix} c_1 \\ \vdots \\ c_G \end{pmatrix}$
24: **return** (M, s)
25: **procedure** $\mathsf{AGOMR3.Retrieve}(\mathsf{pp}, \mathsf{BB}, \mathsf{pk}_\mathsf{det} = (\mathsf{pk}_\mathsf{bfv}, \mathsf{ct}_\mathsf{ID}, \mathsf{ct}_\mathsf{SWK}), \bar{k})$
26: For all $i \in [|\mathsf{BB}|]$, load (M_i, s_i)
27: Use s_i to sample Z_i from $\mathbb{Z}_t^{I\times G'}$
28: Compute $c_i \leftarrow (\mathsf{ct}_\mathsf{ID} \times Z_i) \times \mathsf{M}_i$
29: Proceed as $\mathsf{OMR\text{-}OPT.Retrieve}$ with $(c_i)_{i\in N}$ from line 17
30: **procedure** $\mathsf{AGOMR3.Decode}(\mathsf{pp}, M, \mathsf{sk})$
31: Same as $\mathsf{OMRp2.Decode}$ in [LT22, Algorithm 8]
32: **procedure** $\mathsf{AGOMR3.RegenDetectKey}(\mathsf{pp}, \mathsf{sk} = (\mathsf{sk}_\mathsf{BFV}, \mathsf{sk}_\mathsf{PVW}, \mathsf{id}))$
33: $(\mathsf{sk}'_\mathsf{BFV}, \mathsf{pk}'_\mathsf{BFV}) \leftarrow \mathsf{BFV.KeyGen}(\mathsf{pp}_\mathsf{BFV})$
34: $\mathsf{ct}_\mathsf{SWK}' \leftarrow \mathsf{BFV.Enc}(\mathsf{pk}'_\mathsf{BFV}, \mathsf{sk}_\mathsf{PVW})$
35: $\mathsf{ct}_\mathsf{ID}' \leftarrow \mathsf{BFV.Enc}(\mathsf{pk}'_\mathsf{BFV}, \mathsf{id})$
36: **return** $(\mathsf{sk}' = (\mathsf{sk}'_\mathsf{BFV}, \mathsf{sk}_\mathsf{PVW}, \mathsf{id}), \mathsf{pk}_\mathsf{det} = (\mathsf{pk}'_\mathsf{BFV}, \mathsf{ct}_\mathsf{ID}', \mathsf{ct}_\mathsf{SWK}')$

H_1: The challenger, after seeing Y_0, Y_1 and $Z = Y_0 \cap Y_1$ sent from the adversary, sends $(\mathsf{sk}_i)_{i\in Z}$ to the adversary, and receives $Z' \subseteq Z, (\mathsf{sk}_i, \mathsf{pk}_i)_{i\in Z'}$ back. Then, the challenger draws $b \leftarrow \{0, 1\}$ to select Y_b. Then, it generates PVW ciphertexts $\mathsf{ct}_i \in \mathbb{Z}_t^{n+\ell}$ using pk_i for all $\boldsymbol{i \in Z}$ **rather than all of** $\boldsymbol{Y_b}$ (where the keys in Z' are overwritten by the adversary). **It then samples the remaining $\mathbf{ct}_{\boldsymbol{i}}$ for $\boldsymbol{i \in Y_b \setminus Z}$ uniformly random from $\mathbb{Z}_{\boldsymbol{t}}^{\boldsymbol{n+\ell}}$.** Form the matrix $\tilde{\mathsf{id}}$ with the id_i of each recipients $i \in Y_b$ as on line 22, the challenger computes the matrix M using $\tilde{\mathsf{id}}$ and $(\mathsf{ct}_i)_{i\in[G]}$ as on line 23. Finally, the challenger sends back M and the seed s to compress the ids.

H_0 is indistinguishable from H_1 because they differ only in how ct_i for $i \in Y_b \setminus Z$ are generated, and H_0 generating them by PVW encryption (using honestly generated public keys for which the secret key is unknown and not controlled by the adversary) is computationally indistinguishable from H_1 sampling them uniformly.

Finally, in H_1, since all of ct_i for $i \in Y_b \setminus Z$ are independent of b and $\tilde{\mathsf{id}}$ is full rank (with $1 - \mathsf{negl}(\lambda)$ probability), as in line 23 in Algorithm 3, the adversary cannot do better other than randomly guessing b. Therefore, the adversary cannot break H_0 either (assuming the hardness of LWE).

Compactness: The digest size is almost identical to that of OMR-OPT. The only difference is that now the BFV circuit has $O(G)$ more additions at the same level, which means that the error grows $\log(G)$ and thus to accommodate this error grows, we need to set ring dimension D to grow $\log(G)$ as well. Therefore, the digest size has an extra $\log(G)$ factor.

However, note that this issue can be resolved by a key-switching technique to reduce the ciphertext size. For simplicity, we do not include this technique. In practice, this small error growth does not contribute to the change in digest size at all. See Section 9 for details.

Detection-key-unlinkability: $\mathsf{pk}_{\mathsf{BFV}}$ is simply a fresh BFV public key. $\mathsf{ct}_{\mathsf{ID}}, \mathsf{ct}_{\mathsf{SWK}}$ are both BFV ciphertexts. Thus, by the security of BFV, all these are computationally indistinguishable from other detection keys generated freshly using KeyGen with different sk. □

Amortization and preprocessing. To further reduce the concrete cost, we take some of the computation offline. A plaintext-by-ciphertext multiplication in BFV contains three components: (1) NTT transformation of the BFV ciphertext (resulting in the "NTT form" of the ciphertext); (2) NTT transformation of the plaintext; (3) multiplication between the two NTT form components. Costs are in descending order. We try to reduce the number the first two operations, and amortize the costs over different recipients in the following way:

- Notice that when performing the linear function evaluation (i.e., matrix multiplication), multiple plaintexts generated based on M_s's are multiplied with the same ciphertext (i.e., encrypted id). Thus, we cache the NTT form of the encrypted id to reduce the number of the first type of operation.
- The plaintext NTT transformations for all published M's are shared by all recipients; therefore, the runtime of this part can be amortized over all recipients registered under the same detector.

6.3 Extension to Distinct Payloads

In some applications, a sender addresses a group of recipients with different payloads in a single message (e.g., a Zcash transaction may have multiple recipients, each of which cares only about the data describing the output note they can spend). Our AGOMR definition (Definition 4.1) can be extended to fit this multiple payload setting, where the correctness, privacy, and compactness definitions remain exactly the same, and the soundness extends in the most natural way (i.e., payloads not intended for the recipient should be excluded w.h.p.).

Note that this is similar to OMR (as now each group member gets a different message), but an algorithm for AGOMR with different payloads can exploit the fact that the senders are able to batch the recipients into an ad-hoc group, making the detection process more efficient: for our construction, the detector only has to do one homomorphic PVW decryption and compression.

First, notice that this extension can be trivially implemented through our AGOMR (or OMR) scheme by concatenating all the payloads to a giant payload. However, this (1) increases the digest size from PL to $G \cdot \mathsf{PL}$ (which would break the compactness), (2) the soundness guarantee is violated, as impertinent payloads that are concatenated with the pertinent ones.

The following modification to $\mathsf{AGOMR3}$ resolves this issues and achieves compact multi-payload AGOMR.

- Similarly to the linear function interpolation performed for different clues, with G distinct payloads $(x_1, \ldots, x_G) \in \mathcal{P}^G$ intended for G recipients with id's $(\mathsf{id}_1, \ldots, \mathsf{id}_G) \in \mathcal{D}^G$, the sender interpolates a function $h : \mathcal{D} \to \mathcal{P}$, such that $h(\mathsf{id}_i) = x_i$ for $i \in [G]$. We represent function h as a matrix for the linear transformation from id_i to payload x_i.
- The detector uses the encrypted id in the detection key to evaluate $h(\mathsf{id})$ homomorphically, and obtains a BFV ciphertext encrypting the payload (if the payload is larger than the plaintext space of a BFV ciphertext, we simply use multiple BFV ciphertexts, and for simplicity, we omit the details of this case). Then, the rest almost follows exactly what [LT22, §7.4] has, except that the payload is now a BFV ciphertext, and thus needs to perform ciphertext multiplications when applying Sparse Random Linear Coding (SRLC, [LT22, Section 6.3]). (Alternatively, one can also view the BFV ciphertext as a plaintext payload and proceed exactly as in [LT22].)

Analysis.

- Correctness and soundness: As long as the function h can be successfully interpolated (guaranteed by the linear independency of the compressed IDs), the correctness of our scheme is guaranteed by the correctness of BFV, which follows the same as in the proof of Theorem 6.4. Soundness also naturally follows from the same analysis.
- Privacy: We now also require that h does not reveal the identity of the recipients. W.l.o.g., we assume all the payloads are drawn from the uniformly random distribution over some payload space $\mathcal{P}$ (if not, we first encrypt the payload through a key-private CPA-secure encryption scheme to make them indistinguishable from random). Privacy is thus guaranteed the same way as the current AGOMR privacy.
- Detector runtime: Asymptotically, the runtime does not change much with such a modification. Concretely speaking, the evaluation of h is just a single level and can be done at a very low BFV multiplicative level (as it is only followed by some linear encoding), its cost is very small compared to the rest of the computation. Although the SRLC part will become slower as it requires ciphertext-by-ciphertext multiplication instead of plaintext-by-ciphertext multiplication, it does not contribute much to the overall detector runtime. Thus the total runtime is only slightly affected. (One could also view the BFV encryption of the payload as a BFV plaintext, and proceed exactly at before, at the cost of payload being enlarged.)
- Digest size: By homomorphically evaluating $h(\mathsf{id})$, the detector will receive a BFV ciphertext encrypting a single payload. Therefore, the digest size remains the same and is independent of G.
- Payload size: The sender publishes the matrix representation of function h as payload, which is of size $\sim (G + \log(\lambda)) \cdot |\mathcal{P}|$ (concretely, for a statistical security parameter of ~ 48, we have $(G + 4) \cdot |\mathcal{P}|$). This is slightly larger than the naive solution, which is of size $G \cdot |\mathcal{P}|$.

7 Fixed Group OMR

We proceed to construct Fixed Group OMR (FGOMR), where recipients form groups in advance, and the senders can subsequently send messages to these fixed groups (or subsets thereof), as defined in Section 4.2. This allows reduced detection cost compared to AGOMR.

7.1 Multi-Recipient Encryption

We achieve FGOMR via Multi-Recipient Encryption (MRE), defined below. At a high level, MRE enables the sender to encrypt multiple messages to multiple recipients at the same time. FGOMR can leverage the encryption function in MRE to generate a single clue for multiple recipients, and the detector then uses the decryption function in MRE to evaluate all clues on the board.

Our MRE definition is adapted from [BBKS07]. We extend it with a key-privacy property, since the adversary should not learn which group the message is addressed to unless a (fully malicious) adversary corrupts the recipients in the group. We also make a couple of minor relaxations compared to [BBKS07] (see Theorem 7.1).

Definition 7.1. (Multi-Recipient Encryption). A Multi-Recipient Encryption scheme has the following PPT algorithms:

- $\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, G, P)$: takes a security parameter λ, the number of recipients G in a group, and the total number of recipients P in the system; outputs a public parameter pp.
- $\mathsf{sk} \leftarrow \mathsf{SKGen}(\mathsf{pp})$: takes a public parameter pp and outputs a secret key sk.
- $\mathsf{pk} \leftarrow \mathsf{PKGen}(\mathsf{pp}, \mathsf{sk})$: takes a public parameter pp, a secret key sk; outputs a public key pk.
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, \vec{\mathsf{pk}}, \vec{m})$: takes a public parameter pp, a vector of up to G public keys $\vec{\mathsf{pk}}$, a vector of messages $\vec{m}, |\vec{m}| = |\vec{\mathsf{pk}}|$; outputs a ciphertext $\mathsf{ct} \in \mathcal{C}$
- $m \leftarrow \mathsf{Dec}(\mathsf{pp}, \mathsf{sk}, \mathsf{ct})$: takes a public parameter pp, a secret key sk, a ciphertext ct; outputs a message m

The scheme must satisfy the following properties:

- (Correctness) Let $\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, G, P)$, for $j \in [P]$, let $\mathsf{sk}_j \leftarrow \mathsf{SKGen}(\mathsf{pp}), \mathsf{pk}_j \leftarrow \mathsf{PKGen}(\mathsf{pp}, \mathsf{sk}_j)$. For any set of recipients $Y \subseteq [P], |Y| \leq G$, and any plaintext vector $(m_j)_{j \in [|Y|]} \in \mathcal{M}^{|Y|}$, for all $i \in [|Y|]$, it holds that:
$$\Pr[\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}_i, \mathsf{Enc}(\mathsf{pp}, (\mathsf{pk}_j)_{j \in Y}, (m_j)_{j \in [|Y|]})) = m_i] \geq 1 - \mathsf{negl}(\lambda)$$
.
- (CPA security) An MRE scheme is CPA secure if for any PPT adversary $\mathcal{A}$, it wins the game in Fig. 7 with probability $\leq 1/2 + \mathsf{negl}(\lambda)$, where $Y, Y'Z$ represent groups of recipients.
- (Key Privacy) An MRE scheme is key private if for any PPT adversary $\mathcal{A}$, it wins the game in Fig. 8 with probability $\leq 1/2 + \mathsf{negl}(\lambda)$, where Y_0, Y_1, Z represent groups of recipients.

Remark 7.1. We adapted this primitive and its CPA security definition from [BBKS07], with the following changes: 1) we introduce key privacy; 2) unlike in [BBKS07], we require G and P to be known when $\mathsf{GenParams}$ is invoked (G can be equal to P as in [BBKS07]); 3) we relax correctness probability to 1 - $\mathsf{negl}(\lambda)$, rather than 1 as in [BBKS07]; 4) we separate KeyGen into SKGen and PKGen, since they are invoked separately in our case; 5) in [BBKS07], the adversary needs to first send the number of key pairs that it wants to maliciously overwrite before receiving all honestly generated public keys. Regarding the last point, notice that if there exists an adversary that breaks the CPA security in [BBKS07], then it trivially breaks ours. Therefore, our definition implies the one in [BBKS07].

Adversary		**Challenger**
		$\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, G, P)$
		$\mathsf{sk}_i \leftarrow \mathsf{SKGen}(\mathsf{pp})$ for $i \in [P]$
Choose groups $Z \subseteq [P], \lvert Z \rvert < G$	$\xleftarrow{(\mathsf{pk}_i)_{i\in[P]}, \mathsf{pp}}$	$\mathsf{pk}_i \leftarrow \mathsf{PKGen}(\mathsf{pp}, \mathsf{sk}_i)$ for $i \in [P]$
	$\xrightarrow{Z}$	
Choose $(\mathsf{sk}_i)_{i\in Z'}, Z' \subseteq Z$ to overwrite	$\xleftarrow{(\mathsf{sk}_i)_{i\in Z}}$	
Choose $Y \subseteq [P] \setminus Z$, where $1 \leq \lvert Y \rvert \leq G - \lvert Z \rvert$		
Choose $m_0, m_1 \in \mathcal{M}^{\lvert Y \rvert}, m \in \mathcal{M}^{\lvert Y' \rvert}$	$\xrightarrow{Z', Y, (\mathsf{sk}_i)_{i\in Z'}, m_0, m_1, m}$	$Y' \leftarrow Z \cup Y$
		$\mathsf{pk}_i \leftarrow \mathsf{PKGen}(\mathsf{pp}, \mathsf{sk}_i)$ for $i \in Z$, overwrite the old ones
		$b \leftarrow\$ \{0, 1\}$
	$\xleftarrow{c}$	$c \leftarrow \mathsf{Enc}(\mathsf{pp}, (\mathsf{pk}_i)_{i\in Y'}, m_b \| m)$
	$\xrightarrow{b'}$	(Adversary wins if $b = b'$)

Figure 7: CPA Security game for Multi-Recipient Encryption.

7.2 Naive Solutions

As a stepping stone, we discuss two naive approaches to constructing MRE schemes based on PVW encryption, and explain why they are inapplicable to our application.

A non-key-private solution. A simple solution (a simplified scheme as in [AHK+22, KKPP20, HKP+21], which have a similar issue) is to use the standard PVW encryption scheme as an MRE scheme. To encrypt to G recipients, simply generate G PVW ciphertexts $(\vec{a}_i, \vec{b}_i)_{i\in[G]}$. Furthermore, all recipients can share the same $\vec{a}$ (i.e., $\vec{a}_i = \vec{a}_j$ for all $i, j \in [G]$), and thus the MRE ciphertext is of the form $(\vec{a}, \vec{b}_1, \ldots, \vec{b}_G)$.

However, for correctness, the recipient needs to know which $\vec{b}_i$ ($i \in [G]$) to use during decryption. To guarantee this, one way to do this is that the sender can simply attach a hash function h that hashes a public key pk into a specific $\vec{b}_i$ (the sender can manually avoid collisions). The recipient then computes $h(\mathsf{pk})$ and finds out the correct $\vec{b}_i$. However, this construction does not satisfy the key privacy requirement, as that the intended recipients do not have collision already leaks information.

An inefficient solution. An alternative way is to let the encryption algorithm encrypts the same message λ times (i.e., the new ciphertext includes λ copies of the ciphertext above without $h(\mathsf{pk})$), where λ is the security parameter. Only if the recipient uses the correct key to decrypt, all the λ copies decrypt into the same message, except with $O(\exp(-\lambda))$ probability, by the wrong-key decryption property of PVW [LT22, Def 5.1]. The drawback is that all costs grow with λ.

Moreover, both solutions are not FHE friendly (the first one needs indexing, and the second one requires an equality check), and thus are hard to incorporate into our application.

If an application does not require key unlinkability for its FGOMR, the hash function can be evaluated directly in plaintext, in which case the first solution above is sufficient and more efficient than the key-private MRE below.

Adversary		**Challenger**
		$\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, G, P)$
		$\mathsf{sk}_i \leftarrow \mathsf{SKGen}(\mathsf{pp})$ for $i \in [P]$
Choose groups $Y_0, Y_1 \subseteq [P]$ where $\lvert Y_0\rvert, \lvert Y_1\rvert \leq G$	$\xleftarrow{(\mathsf{pk}_i)_{i\in[P]}, \mathsf{pp}}$	$\mathsf{pk}_i \leftarrow \mathsf{PKGen}(\mathsf{pp}, \mathsf{sk}_i)$ for $i \in [P]$
Let $Z \leftarrow Y_0 \cap Y_1$	$\xrightarrow{Z}$	
Choose $(\mathsf{sk}_i)_{i\in Z'}, Z' \subseteq Z$ to overwrite	$\xleftarrow{(\mathsf{sk}_i)_{i\in Z}}$	
Choose a message vector $m \in \mathcal{M}^G$	$\xrightarrow{Y_0, Y_1, Z', (\mathsf{sk}_i)_{i\in Z'}, m}$	$b \leftarrow_\$ \{0,1\}$
		$\mathsf{pk}_i \leftarrow \mathsf{PKGen}(\mathsf{pp}, \mathsf{sk}_i)$ **for** $i \in Z$, overwrite the old ones
	$\xleftarrow{c}$	$c \leftarrow \mathsf{Enc}(\mathsf{pp}, (\mathsf{pk}_i)_{i\in Y_b}, m[1 : \lvert Y_b\rvert])$
	$\xrightarrow{b'}$	(Adversary wins if $b = b'$)

Figure 8: Key Privacy game for Multi-Recipient Encryption.

7.3 Efficient Key-private MRE

We introduce our main Multi-Recipient Encryption construction, which will be used to construct our main FGOMR scheme in Section 7.4.

High-level Idea. Our starting point is PVW encryption as in [PVW08], and we use the same parameters (n, t, w, ℓ, σ). For simplicity, we start with $\ell = 1$, i.e., given a group of G recipients, the plaintext is represented by $\vec{m} \in \mathbb{Z}_2^G$, one bit per recipient.

Given the group members' secret keys sk_i, ideally we want to find a clue ciphertext $(\vec{a}, \beta)$ such that $\langle \vec{a}, \mathsf{sk}_i \rangle \approx \beta$ for all group member i. However, this seems infeasible (by the Snake-Eye Conjecture [LT22]). Instead, we choose (A, b) pseudorandomly, and have each recipient publish an extended public key that is a function of $(A, \vec{b})$ and sk_i, such that the sender can use these to sample $\vec{a}, \beta$ as a subset-sum of $(A, \vec{b})$ and then efficiently find alpha such that the extended clue ciphertext $(\vec{a}, \alpha, \beta)$ fulfills $\langle a||\alpha, \mathsf{sk}_i \rangle \approx \beta$.

With this intuition in mind, we describe our construction in more detail below.

$\mathsf{GenParams}$: The scheme sets up the public parameters pp according to the PVW security and other probability analysis similar to Section 6.2. The public parameters pp also include a random seed s, from which each recipient in the group can generate a matrix $A \in \mathbb{Z}_t^{w\times n}$ and a vector $\vec{b} \in \mathbb{Z}_t^w$.

SKGen: Each recipient generates their secret keys as a random vector $\mathsf{sk} = \mathsf{ck}||\mathsf{dk} \in \mathbb{Z}_t^{n+L}$, where $\mathsf{ck} \leftarrow_\$ \mathbb{Z}_t^n$ and $\mathsf{dk} \leftarrow_\$ \mathbb{Z}_t^L$. $L = G + O(\lambda)$ is chosen by a similar procedure as G' in Section 6.2.

PKGen: The recipient then computes $\vec{b}' \leftarrow \mathsf{ck}A^\intercal + \vec{e}$, where $\vec{e}$ is a Gaussian noise vector, and publishes $\mathsf{pk} = (\vec{b}', \mathsf{dk})$.

Enc: A sender first draws $\vec{e} \leftarrow_\$ \{0,1\}^w$ and computes $\vec{a} \leftarrow \vec{e}A \in \mathbb{Z}_t^n$, $\beta \leftarrow \langle \vec{e}, \vec{b} \rangle \in \mathbb{Z}_t$ and $\beta'_i = \langle \vec{e}, \vec{b}'_i \rangle \in \mathbb{Z}_t$, for all $i \in [G]$. The sender then finds $\vec{\gamma} \in \mathbb{Z}_t^L$ such that $\langle \vec{\gamma}, \mathsf{dk}_i \rangle = \beta - \beta'_i$ for all $i \in [G]$. This is done by solving $\vec{\gamma}\mathsf{dk_{all}}^\intercal = \vec{\beta}''$, where $\mathsf{dk_{all}} = \begin{pmatrix} \mathsf{dk}_1 \\ \vdots \\ \mathsf{dk}_G \end{pmatrix} \in \mathbb{Z}_t^{G\times L}$, $\vec{\beta}'' = (\beta - \beta'_1 + m[1] \cdot \lceil t/2 \rceil)|| \ldots ||(\beta - \beta'_G + m[G] \cdot \lceil t/2 \rceil) \in \mathbb{Z}_t^G$. The equation system is solvable as long as $\mathsf{dk_{all}}$ has full row rank. The ciphertext is $(\vec{a}, \vec{\gamma}, \beta)$.

Dec: The recipient uses its own secret key $\mathsf{sk} = \mathsf{ck}\|\mathsf{dk}$ to check $|\beta - \langle \vec{a}||\vec{\gamma}, \mathsf{sk} \rangle| \geq \lceil t/4 \rceil$.

To make the process above resistant to maliciously chosen recipients, we again extends the size of dk to be long enough. Similarly to Section 6.2, we let ϵ_{DI} denote that the probability that $\mathsf{dk_{all}}$ is not full row rank, where $\mathsf{dk_{all}} \leftarrow \begin{pmatrix} \mathsf{dk}_1 \\ \vdots \\ \mathsf{dk}_G \end{pmatrix} \in \mathbb{Z}_t^{G\times L}$, and let ϵ_{DS} denote the probability that $\mathsf{dk_{all}} \times Z$ is not full row rank given that $\mathsf{dk_{all}}$ is full row rank, where $Z \leftarrow_{\mathsf{s}'} \mathbb{Z}_t^{L\times G'}$ is generated from a random seed s'. We can efficiently achieve $\epsilon_{\mathrm{DI}}, \epsilon_{\mathrm{DS}} \leq \mathsf{negl}(\lambda)$ as shown in the lemmas in Section 6.2. As before, the sender then solves for $\vec{\alpha} \in \mathbb{Z}_t^{G'}$, such that $\vec{\alpha}(\mathsf{dk_{all}} \times Z)^\intercal = \vec{\beta}''$. The seed s' is appended to the ciphertext.

To extend this idea to $\ell > 1$, we simply repeat the above procedure with ℓ different secret keys sk as in the PVW encryption [PVW08]. We use a single dk instead of having ℓ different dk for better efficiency (also using the same $(\vec{a}, \beta)$). The resulting ciphertext is of form $(\vec{a}, \alpha_1, \ldots, \alpha_\ell, \beta, \mathsf{s}') \in \mathbb{Z}_t^n \times \mathbb{Z}_t^{G'} \times \ldots \times \mathbb{Z}_t^{G'} \times \mathbb{Z}_t \times \mathbb{Z}_{2^\lambda}$, such that for each intended recipient with $(\mathsf{ck}_j, \mathsf{dk})_{j\in[\ell]}$, $\langle \vec{a} || \alpha_j, \mathsf{ck}_j || (\mathsf{dk} \times Z)\rangle \approx \beta$, where $Z \leftarrow_{\mathsf{s}} \mathbb{Z}_t^{L\times G'}$, for all $j \in [\ell]$.

Security follows from the following. In the ciphertext, the distribution of $\vec{a}, \beta, \mathsf{s}'$ are statistically indistinguishable from uniform and are independent. Then, the distribution of $\vec{b}'$ is computationally indistinguishable from uniform by the hardness of LWE, and $\vec{\beta}'$ is computationally indistinguishable from uniform by leftover hash lemma. Thus, the distribution of $\vec{\alpha}$ is computationally indistinguishable from uniform.

Algorithm 4 provides the full construction of our MRE construction, including some details omitted above for simplicity.

Theorem 7.2. For any $P = \mathsf{poly}(\lambda), G = \mathsf{poly}(\lambda)$, assuming hardness of LWE, the scheme MRE in Algorithm 4 is a key-private MRE scheme.

Proof sketch. Correctness: Similarly to the proof of correctness in Theorem 6.4, the probability that the equation system is unsolvable has probability $\mathsf{negl}(\lambda)$. Therefore, we have $A\mathsf{sk} \approx \vec{b}$ and the correctness trivially follows.

CPA security: We prove CPA security by showing that given pp for all $Y \in [P]$ and $|Y| = G$, given $(\mathsf{pk}_i)_{i\in Y}$, for all $\vec{m} \in \mathcal{M}^{|Y|}$, it holds that $\mathsf{ct} = (\vec{a}, \alpha_1, \ldots, \alpha_\ell, \beta, \mathsf{s}') \leftarrow \mathsf{Enc}(\mathsf{pp}, (\mathsf{pk}_i)_{i\in Y}, \vec{m})$ is computationally indisinguishable from a uniformly-drawn tuple $u \leftarrow_\$ \mathbb{Z}_t^n \times \mathbb{Z}_t^{G'} \times \ldots \times \mathbb{Z}_t^{G'} \times \mathbb{Z}_t \times \mathbb{Z}_{2^\lambda}$. First, s' is a random seed, and $\vec{a}, \beta$ are statistically indistinguishable from random vectors trivially. The only thing to prove is that $(\alpha_i)_{i\in[\ell]}$ is computationally indistinguishable from random vectors.

Notice that for each $\alpha_{i\in[\ell]}$, it is generated by solving equation $\vec{\alpha}\mathsf{XK}^\intercal = \beta''$ on line 24 in Algorithm 4. Thus, $(\alpha_i)_{i\in[\ell]}$ is indistinguishable from a random vector following from the fact that β'' is indistinguishable from a random vector by the hardness of LWE assumption.

Since the adversary can only choose the randomness used to generate the malicious keys, the analysis remains the same for those corrupted keys (as the indistinguishability result for other keys remain).

Such indistinguishability implies CPA security, as if there exists a PPT adversary who can break the CPA game, it can be used to break such indistinguishability.

Key Privacy: Key privacy is guaranteed as long as the ciphertexts generated by two keys are from (computationally) indistinguishable distributions. This follows the same way argument above for CPA security. □

7.4 Applying MRE to FGOMR

Building FGOMR using our MRE construction MRE is straightforward: given the OMR scheme in Section 5, we replace the PVW secret keys with the keys generated by $\mathsf{MRE.SKGen}$. The group clue key can simply be a vector of all the members' public keys generated by $\mathsf{MRE.PKGen}$ ($h(Y)$ is used as the shared randomness, where h is a random oracle and Y is the group of recipients). During the clue generation, instead of using PVW encryption, we use the MRE scheme to encrypt ℓ ones and proceed with the rest as the original OMR scheme. The detector now homomorphically decrypts an MRE ciphertext, instead of a PVW ciphertext, using $(\mathsf{ck}, \mathsf{dk})$, encrypted under BFV public keys, sent in the detection key.

Detection-key-unlinkability. Similarly to the id-based AGOMR construction in Section 6.2, we encrypt dk using BFV. $\mathsf{RegenDetectKey}$, by generating new BFV public keys and encrypt ck and dk accordingly.

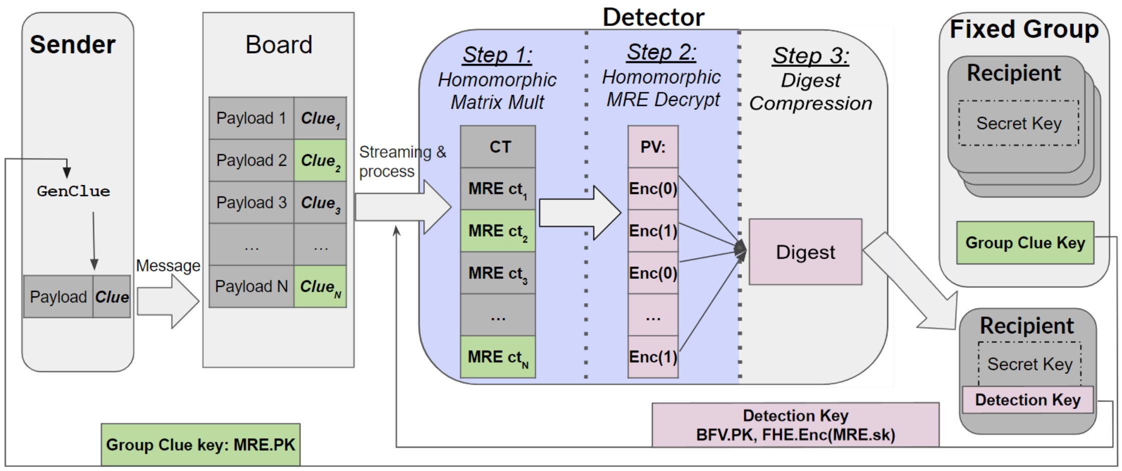


Figure 9: Main components of our Fixed Group OMR construction.

The full algorithm is given in Algorithm 5. A high-level overview is given in Fig. 9. Compared to Fig. 5, the clues remain to be encryptions of zeros, but instead of PVW encryption, we have MRE encryption. Thus, step 2 also has small changes.

Theorem 7.3. The scheme $\mathsf{FGOMR1}$ in Algorithm 5 is a FGOMR scheme for $N < D \cdot t/2$, $P = \mathsf{poly}(\lambda), T = \mathsf{poly}(\lambda)$ assuming security of MRE encryption and security of BFV leveled HE, when instantiated with PRF f and an SRLC scheme SRLC (Definition B.1). Moreover when instantiated with $\mathsf{SRLC1}$ ([LT22, Alg 3]), $\mathsf{FGOMR1}$ is also compact.

Proof sketch. Completeness: Follows from the correctness of $\mathsf{OMR\text{-}OPT}$ and the correctness of the underlying MRE scheme.

Soundness: Similarly to Theorem 6.4 soundness proof.

Computational privacy: Proof same as the MRE scheme we have except that now each group has a shared random matrix A (also uniformly (pseudo-)randomly sampled), but the argument is not affected.

Compactness: Exactly the same as $\mathsf{OMR\text{-}OPT}$ digest size.

Detection-key-unlinkability: Same argument as the detection-key-unlinkability argument in the proof of Theorem 6.4.

□

Alternating the order of the matrix multiplications. For better performance, we introduce another optimization. Since the entire decryption is under FHE, we would have a homomorphic matrix multiplication to calculate $\mathsf{dk} \times Z$ at Algorithm 4 line 42 , which is of size $O(G^2 \log(P))$. Instead, we first perform $\vec{\alpha_i}' \leftarrow \vec{\alpha_i} \times Z^\intercal$ for all $i \in [\ell]$, and then compute $\langle \vec{a} || \vec{\alpha_i}', \mathsf{ck}_i || \mathsf{dk} \rangle$ under FHE. The latter operation is an inner product evaluated homomorphically, which is only of size $O(G \log(P)\ell)$. Since ℓ is asymptotically $\sim \log(\lambda)$ and practically with a very small constant (e.g., 4 as in [LT22, Section 10]), the runtime will be both asymptotically and concretely faster. [8].

Remark 7.4. As in Theorem 6.2, if we restrict the groups to be honestly formed, we do not need to consider the union bound in ϵ_{DI}, which is the probability of existing a combination of G id's among P recipients to be linearly dependent, but instead, only evaluate the probability of a random group of size G to be linearly independent. Thus, the size of L can be reduced from $O(G \log P)$ to $G + O(\lambda)$. The number of homomorphic

[8] A similar technique can be applied for our AGOMR construction. However, that only changes the matrix multiplication size from $O(G^2 \log(P) + G(n+\ell))$ to $O(G \log(P)(n+\ell))$, which is not necessarily an improvement both asymptotically and concretely

Algorithm 4 MRE : Group Encryption

procedure $\mathsf{MRE.GenParams}(\lambda, G, p)$
  Choose (n, w, ℓ, t, σ) according to PVW security and let $\mathsf{pp}_{\mathsf{PVW}} \leftarrow (n, w, \ell, t, \sigma)$
  If $t \leq G^2$, choose $t = G^2$ and $\mathsf{pp}_{\mathsf{PVW}}$ accordingly.
  Choose the smallest L such that $\binom{P}{G} \cdot \prod_{j=L-G-1}^{L}(1 - 1/t^j) \leq 2^{-\lambda}$.
  Choose the smallest $G' \geq G$ such that $\prod_{i=G'-G-1}^{G'}(1 - 1/t^i) \leq 2^{-\lambda}$.
  Generate a random seed s.
  return $\mathsf{pp} = (\lambda, \mathsf{pp}_{\mathsf{PVW}}, L, G, G', s)$
procedure $\mathsf{MRE.SKGen}(\mathsf{pp} = (\lambda, \mathsf{pp}_{\mathsf{PVW}} = (n, w, \ell, t, \sigma), L, G, G', s))$
  Choose a secret key $\mathsf{ck} = (\mathsf{ck}_1, \ldots, \mathsf{ck}_\ell) \leftarrow \mathbb{Z}_t^{n \times \ell}$ uniformly at random.
  Choose an auxiliary key $\mathsf{dk} \leftarrow \mathbb{Z}_t^L$ uniformly at random.
  return $(\mathsf{ck}, \mathsf{dk})$
procedure $\mathsf{MRE.PKGen}(\mathsf{pp} = (\lambda, \mathsf{pp}_{\mathsf{PVW}} = (n, w, \ell, t, \sigma), L, G, G', s), (\mathsf{ck}, \mathsf{dk}))$
  Use s to randomly generate w vectors $A = \{\vec{a_i} \in \mathbb{Z}_t^n\}_{i \in [w]}$
  Use s to randomly generate w vectors $\vec{b} = \{\vec{b_i} \in \mathbb{Z}_t^\ell\}_{i \in [w]}$
  for $i = 1 \to w$ **do**
    for $l = 1 \to \ell$ **do**
      $b'_{i,l} \leftarrow b_{i,l} - \langle \vec{a}_i, \mathsf{ck}_l \rangle - e$ where e is from some Gaussian distribution χ_σ.
  return $\mathsf{pk} = ((b'_{i,l})_{i \in [w], l \in [\ell]}, \mathsf{dk})$
procedure $\mathsf{EncAux}(\mathsf{pp} = (\lambda, \mathsf{pp}_{\mathsf{PVW}}, L, G, G', s), (\vec{a}, \beta), (\beta'_j, \mathsf{dk}_j)_{j \in [G]}, \mathsf{s}')$
  $\mathsf{dk}_{\mathsf{all}} \leftarrow \begin{pmatrix} \mathsf{dk}_1 \\ \vdots \\ \mathsf{dk}_G \end{pmatrix} \in \mathbb{Z}_t^{G \times L}$
  $Z \leftarrow_{\mathsf{s}'} \mathbb{Z}_t^{L \times G'}$
  $\mathsf{XK} = \mathsf{dk}_{\mathsf{all}} \times Z, \mathsf{XK} \in \mathbb{Z}_t^{G \times G'}$
  $\beta'' = \beta'_1 \| \cdots \| \beta'_G \in \mathbb{Z}_t^G$
  Solve the linear equation system $\alpha \mathsf{XK}^\intercal = \vec{\beta''}$ and let $\alpha \leftarrow_\$ \mathbb{Z}_t^{G'}$ if the equation system is underdetermined
  return α
procedure $\mathsf{MRE.Enc}(\mathsf{pp} = (\lambda, \mathsf{pp}_{\mathsf{PVW}}, L, G, G', s), \{\mathsf{pk}_j = ((b'_{i,l,j})_{i \in [w], l \in [\ell]}, \mathsf{dk}_j)\}_{j \in [G]}, \vec{m} \in \mathbb{Z}_t^\ell)$
  ▷ To address less than G, pad with dummy parties using $\mathsf{MRE.PKGen}$.
  Use s to generate w vectors $A = \{\vec{a_i} \in \mathbb{Z}_t^n\}_{i \in [w]}$ ▷ Same A as generated by each party
  Use s to generate w vectors $\vec{b} = \{\vec{b_i} \in \mathbb{Z}_t^\ell\}_{i \in [w]}$ ▷ Same B as generated by each party
  Define a vector $\vec{t} = \frac{t}{2} \cdot \vec{m} \in \mathbb{Z}_t^\ell$
  Sample vector $\vec{e} \leftarrow \{0, 1\}^w \in \mathbb{Z}_2^w$ uniformly at random.
  $(\vec{a}, \beta) \leftarrow (\vec{e}A, \vec{e}\vec{b} + \vec{t}) \in \mathbb{Z}_t^n \times \mathbb{Z}_t^\ell$
  for $j = 1$ to G **do**
    $\beta'_{l,j} \leftarrow \vec{e}(b'_{\cdot,l,j}) \in \mathbb{Z}_t^\ell$, for $l \in [\ell]$
  Generate some randomness s'
  for $l = 1$ to ℓ **do**
    $\alpha_l \leftarrow \mathsf{EncAux}(\mathsf{pp}, (\vec{a}, \beta), (\beta'_{l,j}, \mathsf{dk}_j)_{j \in [G]}, \mathsf{s}')$
  $\mathsf{ct} \leftarrow (\vec{a}, \alpha_1, \ldots \alpha_\ell, \beta, \mathsf{s}')$
  return ct
procedure $\mathsf{MRE.Dec}(\mathsf{pp} = (\lambda, \mathsf{pp}_{\mathsf{PVW}}, L, G, G', s), (\mathsf{ck}, \mathsf{dk}), \mathsf{ct} = (\vec{a}, \alpha_1, \ldots \alpha_\ell, \beta, \mathsf{s}'))$
  $Z \leftarrow_{\mathsf{s}'} \mathbb{Z}_t^{L \times G'}$ ▷ Same Z as generated during encryption
  $\mathsf{dk}' \leftarrow \mathsf{dk} \times Z$
  $\vec{d} = \vec{b} - (\langle \vec{a} || \alpha_1, \mathsf{ck}_1 || \mathsf{dk}' \rangle, \ldots, \langle \vec{a} || \alpha_\ell, \mathsf{ck}_\ell || \mathsf{dk}' \rangle) \in \mathbb{Z}_t^\ell$
  $\vec{m} = \lfloor \frac{\vec{d} + t/4}{t/2} \rceil \in \mathbb{Z}_2^\ell$

Algorithm 5 FGOMR1: Compact Fixed Group Oblivious Message Retrieval

procedure $\mathsf{FGOMR1.GenParams}(1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n}, G, P)$
    Find some secure MRE parameter $\mathsf{pp_{MRE}}$ accordingly to $\mathsf{MRE.GenParams}$ in Algorithm 4.
    Find some secure BFV parameter $\mathsf{pp_{BFV}}$ as in [LT22]
    return $\mathsf{pp} = (1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n}, G, \mathsf{pp_{MRE}}, \mathsf{pp_{BFV}})$
procedure $\mathsf{FGOMR1.PersonalKeyGen}(\mathsf{pp} = (1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n}, G, \mathsf{pp_{MRE}}, \mathsf{pp_{BFV}}))$
    $(\mathsf{ck_{MRE}}, \mathsf{dk}) \leftarrow \mathsf{MRE.SKGen}(\mathsf{pp_{MRE}})$
    $(\mathsf{sk_{BFV}}, \mathsf{pk_{BFV}}) \leftarrow \mathsf{BFV.KeyGen}(\mathsf{pp_{BFV}})$
    $\mathsf{ct_{sk}} \leftarrow \mathsf{BFV.Enc}(\mathsf{pk_{BFV}}, \mathsf{ck_{MRE}})$
    $\mathsf{ct_{dk}} \leftarrow \mathsf{BFV.Enc}(\mathsf{pk_{BFV}}, \mathsf{dk})$
    return $(\mathsf{sk} = (\mathsf{sk_{BFV}}, \mathsf{ck_{MRE}}, \mathsf{dk}), \mathsf{pk_{det}} = (\mathsf{pk_{BFV}}, \mathsf{ct_{sk}}, \mathsf{ct_{dk}})))$
procedure $\mathsf{FGOMR1.GroupKeyGenAux}(\mathsf{pp} = (1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n}, G, \mathsf{pp_{MRE}}, \mathsf{pp_{BFV}}), \mathsf{sk}, Y)$
    $\mathsf{r} \leftarrow h(Y)$ ▷ h is a random oracle
    Replace the r in $\mathsf{pp_{PVW}}$ with this new s
    $((b'_i)_{i\in[w]}, \mathsf{dk}) \leftarrow \mathsf{MRE.PKGen}(\mathsf{pp_{MRE}}, \mathsf{sk})$
    $\mathsf{gPKshare} = (\mathsf{r}, (b'_i)_{i\in[w]}, \mathsf{dk})$
    return $\mathsf{gPKshare}$
procedure $\mathsf{FGOMR1.GroupKeyGen}(\mathsf{pp}, \left(\mathsf{gPKshare}_j = (\mathsf{r}, (b'_{i,j})_{i\in[w]}, \mathsf{dk}_j)\right)_{j\in[G]})$
    ▷ To address less than G, pad with dummy parties using FGOMR1.GroupKeyGenAux.
    return $\mathsf{pk_{clue}} = \left(\mathsf{r}, (b'_{i,j})_{i\in[w]}, \mathsf{dk}_j\right)_{j\in[G]}$
procedure $\mathsf{FGOMR1.GenClue}(\mathsf{pp} = (1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n}, G, \mathsf{pp_{MRE}}, \mathsf{pp_{BFV}}), \mathsf{pk_{clue}} = \left(\mathsf{r}, (b'_{i,j})_{i\in[w]}, \mathsf{dk}_j\right)_{j\in[G]}, Y', x)$
    $\mathsf{pk_{clue}}' \leftarrow \left(\mathsf{r}, (b'_{i,j})_{i\in[w]}, \mathsf{dk}_j\right)_{j\in Y'}$
    $c \leftarrow \mathsf{MRE.Enc}(\mathsf{pp_{MRE}}, \mathsf{pk_{clue}}', 1^\ell)$
    return (x, c)
procedure $\mathsf{FGOMR1.Retrieve}(\mathsf{pp}, \mathsf{BB}, \mathsf{pk_{det}} = (\mathsf{pk_{BFV}}, \mathsf{ct_{sk}}, \mathsf{ct_{dk}}), \bar{k})$
    for $i \in [|\mathsf{BB}|]$ **do**
        Homomorphically perform $\mathsf{MRE.Dec}(\mathsf{pp_{MRE}}, (\mathsf{ck_{MRE}}, \mathsf{dk}), \mathsf{BB}.c_i)$ using BFV
            ▷ The decryption process is almost the same, so the techniques in OMR-OPT can be reused.
    Proceed as line 35 in OMR-OPT.
procedure $\mathsf{FGOMR1.Decode}(\mathsf{pp}, M, \mathsf{sk})$
    Same as OMR-OPT.Decode
procedure $\mathsf{FGOMR1.RegenDetectKey}(\mathsf{pp}, \mathsf{sk} = (\mathsf{sk_{BFV}}, \mathsf{ck_{MRE}}, \mathsf{dk}))$
    $(\mathsf{sk}'_\mathsf{BFV}, \mathsf{pk}'_\mathsf{BFV}) \leftarrow \mathsf{BFV.KeyGen}(\mathsf{pp_{BFV}})$
    $\mathsf{ct}'_\mathsf{sk} \leftarrow \mathsf{BFV.Enc}(\mathsf{pk}'_\mathsf{BFV}, \mathsf{ck_{MRE}})$
    $\mathsf{ct}'_\mathsf{dk} \leftarrow \mathsf{BFV.Enc}(\mathsf{pk}'_\mathsf{BFV}, \mathsf{dk})$
    return $(\mathsf{sk}' = (\mathsf{sk}'_\mathsf{BFV}, \mathsf{ck_{MRE}}, \mathsf{dk}), \mathsf{pk_{det}} = (\mathsf{pk}'_\mathsf{BFV}, \mathsf{ct}'_\mathsf{sk}, \mathsf{ct}'_\mathsf{dk})$

operations then reduces from $O(G\log(P)\ell)$ to $O(G\ell)$, which gives a better asymptotic detector runtime. We call this variant FGOMR2.

7.5 Applying FGOMR to AGOMR

An interesting observation is that we can extend our FGOMR scheme to AGOMR by forcing all recipients to share the same $A, \vec{b}$ in their public key and treat this pk as the gPKshare in FGOMR Definition 4.4. Then, everything follows: senders can arbitrarily choose up to G recipients as a subgroup of all P recipients to send a message.

Since we treat all P recipients as a whole group, the recipients do not need to generate different gPKshare for each different group, and thus the efficiency will match our FGOMR scheme. However, this comes with two caveats: (1) DoS resistance is basically broken, and (2) the privacy guarantee becomes weaker.

Broken DoS resistance. We provide a high-level idea here before formally defining DoS resistance in Section 8: δ-DoS-resistance means that a clue should not be detected as pertinent by more than $\delta - 1$ honest recipients.

When generalizing FGOMR to AGOMR, because all of the honest recipients share the same A, a malicious sender is able to send a single spam message to a large number of recipients $\gg G$ with high probability. In other words, recipient i is publishing some A, B_i such that $A \cdot \mathsf{sk}_i \approx B_i$. Therefore, by crafting some vector e, the adversary can generate eA and check eB_i for all recipients. To spam more than $\delta - 1$ recipients, the adversaries need to find some $\vec{b} \approx B_i e$ for more than δ recipients. This can be done easily by using reject sampling when δ is $O(1)$, which becomes a huge issue for some applications, as discussed in [LT22].

Weaker privacy guarantee than the AGOMR definition. On the other hand, the privacy guarantee of the original AGOMR definition cannot be satisfied. Recall that in Section 4, in AGOMR, for a message sent to G recipients, even if some of the intended recipients have maliciously crafted keys, the identities of other recipients still remain private; while in FGOMR, privacy requires that the intended group does not contain any malicious recipient. Otherwise, that malicious recipient might be able to recover who are the rest of the group. This weaker privacy definition does make sense in the fixed group setting. For example, if a recipient has only joined a single group, it knows that the message must be addressed to that group even though the FGOMR scheme leaks no extra information. Besides, for a recipient joining different groups, it might even become an essential functionality for the recipient to know which group (or, for example, emailing list) this message is addressed to. However, in the ad-hoc setting, this is impractical.

The stronger definition cannot be satisfied because the PVW ciphertexts in our FGOMR construction share the same randomness. By controlling some of the keys, the adversary might be able to recover some or all of the randomness used and thus makes it insecure. However, this has a minor effect as G is usually small, and thus the randomness the adversary can recover is limited. Therefore, it is still possible that with certain parameters, our FGOMR construction can achieve the same level of privacy as our AGOMR construction. We leave the formal argument to future work. A simpler way to resolve this, as we have mentioned in Theorem 4.4, is to use a ZK proof to show that the keys are indeed generated from PersonalKeyGen, as the keys that are semi-maliciously generated cannot break the privacy.

In general, for applications that only require a weaker privacy guarantee or accept the ZK-based solution, and do not need DoS resistance, applying the FGOMR scheme to AGOMR yields better overall performance.

8 DoS Resistance

In a messaging system, an attacker may conduct a Denial of Service (DoS) attack on recipients by simply sending numerous of messages to them, increasing the cost of retrieval and processing. Furthermore, since retrieval queries place a bound on the number of pertinent messages retrieved, an excessive number of new messages eventually causes overflows and prevents access to other messages (as in [MSS+22, LT22]) unless the bound $\bar{k}$ is increased (which necessarily increases the digest size).

Inevitably, an attacker may spam individual recipients with messages, and pay a corresponding linear cost. However, something worse can happen: the attacker might send a message that is detected as pertinent to more than one recipient. In this case, the attacker can cause overflows in many recipients by sending just a small amount of messages. This problem is studied and mitigated in [LT22, §8], for the case of (single-recipient) OMR.

In the group setting, the attacker may inevitably spam *groups* of up to G recipients; thus we want to prevent a worse case (e.g., spamming most users with "wildcard clues").

In this section, we adapt and generalize the DoS resistance definition from [LT22, §8.3] to define a stronger security notion for GOMR. Intuitively, for a message with its corresponding clue maliciously crafted, no more than δ honest recipients should detect it as pertinent. Since by definition, GOMR allows G recipients to be addressed by a single message, by default, $\delta \geq G$, and we want δ to be as close as possible to G. For FGOMR, we define a stronger notion, saying that (honest) recipients in different groups cannot be spammed together.

We then show how to achieve these notions of DoS-resistant AGOMR and FGOMR, under suitable assumptions.

8.1 Ad-hoc Group OMR

To formalize these properties, as in [LT22, §8.3], we define an *indicator* predicate $\mathcal{I}(\mathsf{pp}, x, c, \mathsf{pk}_{\mathsf{clue}}, \mathsf{sk})$ as a ground truth for whether a given message (x, c) is pertinent to a given user specified by one's keys. This predicate, which may not be efficiently computable, should give the natural answer for honestly generated clues. The indicator may answer arbitrarily for otherwise-generated clues, under the restriction that it should claim no more than δ honest recipients as the intended recipients except with a small probability. We define *collision resistance* property for the *indicator* such that it should be difficult for δ recipients to detect the same message as pertinent. In AGOMR, this property prevents a malicious sender from crafting a message to spam more than δ recipients.

Soundness and completeness are then redefined w.r.t the indicator $\mathcal{I}$. As in [LT22, §8.3], to facilitate tight analysis, the completeness bound ϵ_{n} (false negative rate) in the definition is broken up into two components: the rate ϵ_{in} at which the indicator fails to detect truly pertinent messages (which may be non-negligible because an indicator with high thresholds may help achieve collision resistance), and the rate $\epsilon_{\mathrm{n}} - \epsilon_{\mathrm{in}}$ at which the scheme fails to retrieve messages flagged by the indicator (which may be non-negligible because of error sources in the concrete scheme).

Definition 8.1 (DoS-resistant AGOMR)**.** Let AGOMR be an Ad-hoc GOMR scheme with error rates $\epsilon_{\mathrm{n}}, \epsilon_{\mathrm{p}}$ (as in Definition 4.1) and group size upper bound G. An *indicator* for AGOMR, with an *indicator false negative rate* ϵ_{in} (where $\epsilon_{\mathrm{in}} \leq \epsilon_{\mathrm{n}}$) and *collision resistance level* δ, is a function $b \leftarrow \mathcal{I}(\mathsf{pp}, x, c, \mathsf{pk}_{\mathsf{clue}}, \mathsf{sk})$ on public parameter pp, message (x, c), clue key $\mathsf{pk}_{\mathsf{clue}}$, and the corresponding secret key sk that outputs $b \in \{0, 1\}$, such that:

- *(Indicator completeness)* For $\mathsf{pp} \leftarrow \mathsf{AGOMR.GenParams}(1^\lambda, \epsilon_{\mathrm{p}}, \epsilon_{\mathrm{n}}, G, P)$, for any $\tau \in [G]$, honestly-generated key pairs: $(\mathsf{sk}_i, \mathsf{pk}_i = (\mathsf{pk}_{\mathsf{clue}\,i}, \cdot)) \leftarrow \mathsf{AGOMR.KeyGen}(\mathsf{pp})$, $i \in [\tau]$, for any payload x, and honestly-generated clue $c \leftarrow \mathsf{AGOMR.GenClue}(\mathsf{pp}, \mathsf{pk}_{\mathsf{clue}\,1}, \ldots, \mathsf{pk}_{\mathsf{clue}\,\tau}, \mathrm{x})$, it holds that for $i \in [\tau]$:
$$\Pr[\mathcal{I}(\mathsf{pp}, x, c, \mathsf{pk}_{\mathsf{clue}\,i}, \mathsf{sk}_i) = 1] \geq 1 - \epsilon_{\mathrm{in}} - \mathsf{negl}(\lambda) \ .$$
- *(δ-Collision resistance)*For any PPT adversary $\mathcal{A}$, let $\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, \epsilon_{\mathrm{p}}, \epsilon_{\mathrm{n}}, G, P)$, for any δ honestly-generated key pairs: $(\mathsf{sk}_i, \mathsf{pk}_i = (\mathsf{pk}_{\mathsf{clue}\,i}, \cdot) \leftarrow \mathsf{AGOMR.KeyGen}(\mathsf{pp})$, $i \in [\delta]$, and adversarially-generated $(x, c) \leftarrow \mathcal{A}(\mathsf{pk}_1, \ldots, \mathsf{pk}_\delta)$, for $b_i \leftarrow \mathcal{I}(\mathsf{pp}, x, c, \mathsf{pk}_{\mathsf{clue}\,i}, \mathsf{sk}_i)$, it holds that for $i \in [\delta]$:
$$\Pr[b_1 = 1 \wedge \cdots \wedge b_\delta = 1] \leq \epsilon_{\mathrm{p}} + \mathsf{negl}(\lambda) \ .$$

An Ad-hoc GOMR scheme AGOMR is *δ-DoS-resistant* for $\epsilon_{\mathrm{n}}, \epsilon_{\mathrm{p}}$, if there exists an indicator $\mathcal{I}$ with indicator false negative rate ϵ_{in}, DoS resistance level δ, such that for any $G = \mathsf{poly}(\lambda), P = \mathsf{poly}(\lambda)$ and for any PPT adversary $\mathcal{A}$, for $\mathsf{pp} \leftarrow \mathsf{AGOMR.GenParams}(1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n}, G, P)$, $(\mathsf{sk}, \mathsf{pk} = (\mathsf{pk}_{\mathsf{clue}}, \mathsf{pk}_{\mathsf{det}})) \leftarrow \mathsf{AGOMR.KeyGen}(\mathsf{pp})$, 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 $0 < \bar{k} \leq N$, let $M \leftarrow \mathsf{AGOMR.Retrieve}(\mathsf{BB}, \mathsf{pk}_{\mathsf{det}}, \bar{k})$, $\mathsf{PL} \leftarrow \mathsf{AGOMR.Decode}(M, \mathsf{sk})$:

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

Note that this implies the Completeness and Soundness definition in Definition 4.1 analogously to the discussion in [LT22, §8.3] and [LT22, Lem 8.2].

To show our AGOMR scheme is DoS resistant, we need to rely on a generalization of the snake-eye conjecture proposed in [LT22, Conj 8.4] as formalized below.

Conjecture 8.1 (Regev05 is generalized-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 $\delta = \mathsf{poly}(\lambda)$, for any $1 \leq r < q/4$, for key pairs $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda)$, $i \in [\delta+1]$, for ciphertexts $(\vec{a}_i, b_i)_{i\in[\delta+1]} \leftarrow \mathcal{A}($ $(\mathsf{pk})_{i\in[\delta+1]}, r)$, for some coefficients $(c_1, \ldots, c_\delta) \in \mathbb{Z}_q^\delta$, it holds that:

$$\Pr\left[\begin{array}{l} \quad|\langle \vec{a}_1, \mathsf{sk}_1\ \rangle + b_1| \leq r \\ \wedge\ \ldots \\ \wedge\ |\langle \vec{a}_{\delta+1}, \mathsf{sk}_{\delta+1}\rangle + b_{\delta+1}| \leq r \end{array} \wedge\ \exists i \in [\delta+1], c_i\vec{a_i} \neq \vec{0}\ \wedge\ (\vec{a}_{\delta+1}, b_{\delta+1}) = \sum_{i=1}^{\delta} c_i(\vec{a}_i, b_i)\right]\ \leq\ \frac{2r+1}{q} + \mathsf{negl}(\lambda)\ .$$

This generalized snake-eye conjecture is a stronger assumption than the original snake-eye conjecture [LT22, Conj 8.4].

This conjecture can be further generalized form Regev05 encryption to PVW encryption using reject sampling, analogous to [LT22, Lem 8.5], reducing the snake-eye probability from $\frac{2r+1}{q}$ above to $(\frac{2r+1}{q})^\ell$ for $\ell = O(log(\lambda))$ as shown in Theorem 8.2.

Lemma 8.2 (PVW is generalized-snake-eye resistant)**.** Under Conjecture 8.1, for any PPT adversary $\mathcal{A}$, for PVW encryption with modulus q and plaintext space $\mathbb{Z}_2^\ell$, and r such that $(\frac{2r+1}{q})^\ell = \mathsf{poly}(\lambda)$ and remaining parameters for which semantic security hold; for any $\delta = \mathsf{poly}(\lambda)$, $i \in [\delta+1]$, key pairs $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda)$, and ciphertexts $(\vec{a}_i, \vec{b}_i)_{i\in[\delta+1]} \leftarrow \mathcal{A}((\mathsf{pk})_{i\in[\delta+1]}, r)$, let $\vec{m}_j \leftarrow \mathsf{sk}_j^\intercal A + \vec{b}_j$, for some coefficients $(c_1, \ldots, c_\delta) \in \mathbb{Z}_q^\delta$, it holds that:

$$\Pr\left[\begin{array}{c} \forall k \in [\ell] : |m_j[k]| \leq r \\ \wedge\ (\exists i \in [\delta+1], \vec{a}_i \neq \vec{0}) \end{array}\ \wedge\ (\vec{a}_{\delta+1}, \vec{b}_{\delta+1}) = \sum_{i=1}^{\delta} c_i(\vec{a}_i, \vec{b}_i)\right]\ \leq\ (\frac{2r+1}{q})^\ell + \mathsf{negl}(\lambda).$$

Patched AGOMR3. Analogously to [LT22], to prevent trivial DoS attacks, we want to avoid $\mathsf{M_s} \times (\overline{\mathsf{id}} \times Z) = (\vec{0}, \vec{b})$ for more than $\delta - 1$ id's, where $(\mathsf{M_s}, \mathsf{s})$ is the clue in Algorithm 3, and the pseudorandom matrix Z is sampled using randomness s. Thus, the detector rejects the clue when $\mathsf{M} = 0$.

Theorem 8.3. For any $\epsilon_\mathrm{p} > 0, \epsilon_\mathrm{n} > 0, P = \mathsf{poly}(\lambda), G = \mathsf{poly}(\lambda)$, $\mathsf{AGOMR3}$ in Algorithm 3 (patched as above) is a $(G'+1)$-DoS-resistant Ad-hoc Group Oblivious Message Retrieval scheme where G' is as defined in line 5, when instantiated with any PRF f, assuming the hardness of Ring-LWE and Conjecture 8.1.

Proof sketch. The indicator completeness, DoS-completeness, and DoS-soundness are all straightforward since we perform the homomorphic PVW decryption process as in [LT22, §7.4]. The only exception is when $\mathsf{M} = 0$ for honestly generated clues (M, Z). Notice that $\mathsf{M} = 0$ only if $Z \times [\mathsf{id}_i || \ldots, \mathsf{id}_i^G] \times \mathsf{M} = \mathsf{ct}_i = \vec{0}$ for all $i \in [G]$. Thus, the probability of rejecting an honestly generate clue is $\Pr[\mathsf{M} = 0] \leq \Pr[\forall i \in [G], \mathsf{ct}_i = 0] = t^{-nG} = \mathsf{negl}(\lambda)$.

We then prove for $(G'+1)$-collision resistance based on Conjecture 8.1:

Let $G'' = G' + 1$. Recall that a clue is of the form $(\mathsf{M}, Z) \in \mathbb{Z}_t^{G' \times (n+\ell)} \times \mathbb{Z}_t^{I \times G'}$. If a clue is detected as pertinent by G'' honestly recipients, then there exist G'' id's $(\mathsf{id}_1, \ldots, \mathsf{id}_{G''})$, such that for all $i \in [G'']$, $(\vec{a}_i, \vec{b}_i) \leftarrow (\mathsf{id}_i \times Z) \times \mathsf{M}$, it holds that $\Pr[\forall\ i \in [G''],\ \mathsf{PVW.Dec}(\mathsf{sk}_i, (\vec{a}_i, \vec{b}_i)) = 1^\ell] \geq (\frac{2r+1}{q})^\ell + p(\lambda)$ for some non-neglgible function $p(\lambda)$. Furthermore, this also means that there exists $(c_1, \ldots, c_{G'}) \in \mathbb{Z}_q^\delta$ such that $(\vec{a}_{G''}, \vec{b}_{G''}) = \sum_{i=1}^{G'} c_i(\vec{a}_i, \vec{b}_i)$. At least one of the $\vec{a}_i$'s is non-zero, since otherwise $\mathsf{M} = 0$, which, due to the aforementioned patch, would make Algorithm 3 reject this clue. Therefore, it breaks Theorem 8.2 and thus Algorithm 3 is $(G' + 1)$-collision resistance. $\square$

Remark 8.4. Conjecture 8.1 is actually stronger than what we need, as in our AGOMR scheme, the adversary cannot output $\delta+1$ clues that are linearly dependent with arbitrary coefficients. Instead, the linear dependency is predetermined by all the ids and the random matrix Z. Although Z is randomly chosen, it is a seed, and thus it works like a random oracle that the adversary needs to call. Therefore, a weaker conjecture would claim that given certain id's and a random oracle determining the linear dependency, the adversary cannot generate pertinent clues according to that certain linear dependency with $\Pr > \frac{2r+1}{q} + \mathsf{negl}(\lambda)$. Although there are exponentially many possible linear dependencies there (i.e., P choose G possibilities), a PPT adversary may only be able to go through a polynomial amount of them. This weaker conjecture is sufficient for us to prove the collision resistance for Algorithm 3. However, for the readability and simplicity of the paper, we use this stronger conjecture. A possible clean change of underlying conjecture is left to future work.

8.2 Fixed Group OMR

As mentioned, for FGOMR, we capture an even stronger security notion. Essentially, we require that as long as two honest recipients are not in the same group, they should not be spammed at the same time except with a small probability. Since there are at most G honest recipients in each group, this property is strictly stronger than the property defined in Definition 8.1 even when $\delta = G+1$ (which is the best security property that can be achieved for AGOMR).

Definition 8.2 (DoS-resistant FGOMR)**.** Let FGOMR be a Fixed GOMR scheme with error rates $\epsilon_\mathrm{n}, \epsilon_\mathrm{p}$ and group size upper bound G (as in Definition 4.4). An *indicator* with an *indicator false negative rate* $\epsilon_\mathrm{in} \leq \epsilon_\mathrm{n}$ for FGOMR is a function $b \leftarrow \mathcal{I}(\mathsf{pp}, x, c, \{\mathsf{pk_{clue}}\}, \mathsf{sk})$ on public parameter pp, message (x, c), the recipient's secret key sk, clue keys $\{\mathsf{pk_{clue}}\}$ for all groups that include that recipient, that outputs $b \in \{0, 1\}$, such that:

- *(Indicator completeness)* For $\mathsf{pp} \leftarrow \mathsf{FGOMR.GenParams}(1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n}, G, P)$, any $g \leq G$ and any set $Y \subseteq [P]$, $|Y| = g$, and any subset $Y' \subseteq Y$; for honestly generated keys $(\mathsf{sk}_i, \cdot) \leftarrow \mathsf{FGOMR.PersonalKeyGen}(\mathsf{pp})$, $\mathsf{gPKshare}_i \leftarrow \mathsf{FGOMR.GroupKeyGenAux}(\mathsf{pp}, \mathsf{sk}_i, \mathrm{Y})$, $i \in Y$, $\mathsf{pk_{clue}} \leftarrow \mathsf{FGOMR.GroupKeyGen}(\mathsf{pp}, \{\mathsf{gPKshare}_i\}_{i \in Y})$, for any payload x, and honestly generated clue $c \leftarrow \mathsf{FGOMR.GenClue}(\mathsf{pp}, \mathsf{pk_{clue}}, Y', \mathrm{x})$, it holds that for $i \in Y'$:

$$\Pr[\mathcal{I}(\mathsf{pp}, x, c, \{\mathsf{pk_{clue}}\}, \mathsf{sk}_i) = 1] \geq 1 - \epsilon_\mathrm{in} - \mathsf{negl}(\lambda) \ .$$

- *(Collision resistance)* For any PPT adversary $\mathcal{A}$, let $\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, \epsilon_\mathrm{p}, \epsilon_\mathrm{n}, G, P)$, $P = \mathsf{poly}(\lambda)$; with $g, g' \in [G]$, and any two subsets $Y, Y' \subseteq [P]$ with $|Y| = g$, $|Y'| = g'$; for honestly generated keys $(\mathsf{sk}_i, \cdot) \leftarrow \mathsf{FGOMR.PersonalKeyGen}(\mathsf{pp})$,$(\mathsf{sk}'_j, \cdot) \leftarrow \mathsf{FGOMR.PersonalKeyGen}(\mathsf{pp})$,
$\mathsf{gPKshare}_i \leftarrow \mathsf{FGOMR.GroupKeyGenAux}(\mathsf{pp}, \mathsf{sk}_i, \mathrm{Y})$ $(i \in [g])$,
$\mathsf{gPKshare}'_j \leftarrow \mathsf{FGOMR.GroupKeyGenAux}(\mathsf{pp}, \mathsf{sk}'_j, Y')$ $(j \in [g'])$,
$\mathsf{pk_{clue}} \leftarrow \mathsf{FGOMR.GroupKeyGen}(\mathsf{pp}, \mathsf{gPKshare}_1, \ldots, \mathsf{gPKshare}_g)$,
$\mathsf{pk_{clue}}' \leftarrow \mathsf{FGOMR.GroupKeyGen}(\mathsf{pp}, \mathsf{gPKshare}'_1, \ldots, \mathsf{gPKshare}'_{g'})$,
and for adversarially-generated $(x, c) \leftarrow \mathcal{A}(\mathsf{pp}, \{\mathsf{gPKshare}_i\}_{i \in [g]}, \{\mathsf{gPKshare}'_j\}_{j \in [g']}, \mathsf{pk_{clue}}, \mathsf{pk_{clue}}')$; for any $i \in [g], j \in [g']$, let $b \leftarrow \mathcal{I}(\mathsf{pp}, x, c, \mathsf{pk}_{\mathsf{clue}i}, \mathsf{sk}_i)$ and $b' \leftarrow \mathcal{I}(\mathsf{pp}, x, c, \mathsf{pk}'_{\mathsf{clue}j}, \mathsf{sk}'_j)$, it holds that: $\Pr[b = 1 \wedge\ b' = 1] \leq \epsilon_\mathrm{p} + \mathsf{negl}(\lambda)$.

A Fixed GOMR scheme FGOMR is *DoS-resistant* for ϵ_{n}, ϵ_{p}, and G if there exists an indicator $\mathcal{I}$ with an indicator false negative rate ϵ_{in} for FGOMR such that for any $P = \mathsf{poly}(\lambda)$, and any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, for $\mathsf{pp} \leftarrow \mathsf{FGOMR.GenParams}(1^\lambda, \epsilon_{\mathrm{p}}, \epsilon_{\mathrm{n}}, G, P)$, $(\mathsf{sk}_i, \mathsf{pk}_{\mathsf{det}\,i}) \leftarrow \mathsf{FGOMR.PersonalKeyGen}(\mathsf{pp})$ for $i \in [P]$; adversarially generate state and groups $(\mathsf{st}, \{Y_j\}_{j\in[W]}) \leftarrow \mathcal{A}_1(\mathsf{pp})$ where $Y_j \subset [P], |Y_j| \le G$, for any $W = \mathsf{poly}(\lambda)$; for all $j \in [W], i \in Y_j$, $\mathsf{gPKshare}_{i,j} \leftarrow \mathsf{FGOMR.GroupKeyGenAux}(\mathsf{pp}, \mathsf{sk}_i, Y_j)$, $\mathsf{pk}_{\mathsf{clue}\,j} \leftarrow \mathsf{FGOMR.GroupKeyGen}(\mathsf{pp}, \{\mathsf{gPKshare}_i\}_{i\in Y_j})$, and adversarially-generated board $\mathsf{BB} \leftarrow \mathcal{A}_2(\mathsf{pp}, \{\mathsf{gPKshare}_{i,j}\}_{j\in[W], i\in Y_j}, \{\mathsf{pk}_{\mathsf{clue}\,j}\}_{j\in[W]}, \mathsf{st})$ where $\mathsf{BB} = (\,(x_1, c_1), \ldots, (x_N, c_N)\,)$ with unique x_i, for any $0 < \bar{k} \le N$, for $i \in [P]$, let $M \leftarrow \mathsf{Retrieve}(D, \mathsf{pk}_{\mathsf{det}\,u}, \bar{k})$, $\mathsf{PL} \leftarrow \mathsf{Decode}(M, \mathsf{sk})$:

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

Patched FGOMR1. Analogously to AGOMR3, to prevent trivial DoS attacks, we modify our FGOMR1.Retrieve in Algorithm 5 to reject clues $c = (\vec{a} \| \alpha_1 \| \ldots \| \alpha_\ell, \vec{b}, \mathsf{s})$ where $\vec{a} = \vec{0}$.

To prove the DoS-resistance of our FGOMR scheme, we first prove the following lemma:

Lemma 8.5. For any PPT adversary $\mathcal{A}$, for Regev05 encryption with modulus q and plaintext space $\mathbb{Z}_2$, and any $1 \le r \le q/4$ and remaining parameters for which the semantic security holds, for $i \in [2]$, $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda)$, for ciphertexts $(\vec{a}, \vec{b}_i)_{i\in[2]} \leftarrow \mathcal{A}(\mathsf{pk}_1, \mathsf{pk}_2, r)$, it holds that:

$$\Pr\left[\begin{array}{c} |\langle \vec{a}, \mathsf{sk}_1\rangle + b_1| \le r \\ \wedge\ |\langle \vec{a}, \mathsf{sk}_2\rangle + b_2| \le r \end{array} \ \wedge\ \vec{a} \ne \vec{0}\right] \ \le\ \frac{2r+1}{q} + \mathsf{negl}(\lambda)$$

assuming [LT22, Conj 8.4] holds when replacing n there with $n-1$.

Proof sketch. Suppose that we have an adversary $\mathcal{A}$ that breaks this lemma, we construct an adversary $\mathcal{A}'$ that breaks [LT22, Conj 8.4] as follows:

Given $\mathsf{pk} = (A, \vec{b}) \in \mathbb{Z}_q^{w\times(n-1)} \times \mathbb{Z}_q^{w\times 1}$ and $\mathsf{pk}' = (A', \vec{b}') \in \mathbb{Z}_q^{w\times(n-1)} \times \mathbb{Z}_q^{w\times 1}$ and r, $\mathcal{A}'$ generates $\alpha_1, \alpha_2 \leftarrow_\$ \mathbb{Z}_q^{w\times 1}$, $s_1, s_2 \leftarrow_\$ \mathbb{Z}_q$, and computes $\vec{b} \leftarrow \vec{b} + \alpha_1 \times s_1$ and $\vec{b}' \leftarrow \vec{b}' + \alpha_2 \times s_2$. $\mathcal{A}'$ then randomly selects $i \in [n-1]$, and inserts $\alpha_{j\in\{1,2\}}$ to the i-th column, i.e., $A \leftarrow A[0:i]||\alpha_1||A[i:n-1]$ and $A' \leftarrow A'[0:i]||\alpha_2||A'[i:n-1]$.

Then, $\mathcal{A}'$ sends $\mathsf{pk}_1 = (A, \vec{b}), \mathsf{pk}_2 = (A', \vec{b}'), r$ to $\mathcal{A}$. After getting $(\vec{a}, b_1, b_2)$ back, check whether $b_1 + \vec{a}[i]s_1 = b_2 + \vec{a}[i]s_2$. If so, let $\vec{a}' = \vec{a}[0:i-1]||\vec{a}[i+1:n]$, output $(\vec{a}, b_1 + \vec{a}[i]s_1)$. Otherwise, output $\mathsf{Enc}(\mathsf{pk}, 0)$.

Note that trivially, $\mathsf{pk}_1, \mathsf{pk}_2$ are indistinguishable from two honestly generated PVW public keys, and it holds that $|\langle \vec{a}', \mathsf{sk}_1\rangle + b_1 + \vec{a}[i]s_1| \le r$ and $|\langle \vec{a}', \mathsf{sk}_1\rangle + b_2 + \vec{a}[i]s_2| \le r$ with probability $\ge \frac{2r+1}{q} + f(\lambda)$ for some non-negligible function f. Since s_1, s_2 are all chosen randomly and independently and are masked by α_1, α_2 before given to $\mathcal{A}$ to calculate $\vec{a}$, we have $\Pr[b_1 + \vec{a}[i]s_1 = b_2 + \vec{a}[i]s_2] \ge 1/q - \mathsf{negl}(\lambda)$.

For the case where $b_1 + \vec{a}[i]s_1 \ne b_2 + \vec{a}[i]s_2$ and $(\vec{a}, b) \leftarrow \mathsf{Enc}(\mathsf{pk}, 0)$, the probability that $|\langle \vec{a}, \mathsf{sk}\rangle + b| \le r$ and $|\langle \vec{a}, \mathsf{sk}\rangle + b| \le r$ is $\ge \frac{2r+1}{q} - \mathsf{negl}(\lambda)$. Therefore, we have that this $\mathcal{A}'$ breaks the conjecture with probability $\ge \frac{2r+1}{q} + f(\lambda)/q - \mathsf{negl}(\lambda)$, where $f(\lambda)/q$ is non-negligible. $\square$

This lemma can be generalized from Regev05 encryption to PVW encryption using reject sampling, analogous to [LT22, lemma 8.5] as follows, which is then used to prove Theorem 8.7.

Lemma 8.6. For any PPT adversary $\mathcal{A}$, for PVW encryption with modulus q and plaintext space $\mathbb{Z}_2^\ell$, and r such that $(\frac{2r+1}{q})^\ell = \mathsf{poly}(\lambda)$ and remaining parameters for which the semantic security holds, for $i \in [2]$, $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda)$, for ciphertext $(\vec{a}, (\vec{b}_i)_{i\in[2]}) \leftarrow \mathcal{A}(\mathsf{pk}_1, \mathsf{pk}_2, r)$, let $\vec{m}_{i\in[2]} \leftarrow \mathsf{sk}_i^\intercal \vec{a} + \vec{b}_i$, it holds that:

$$\Pr\Big[(\,\forall k \in [\ell] : |m_1[k]| \le r \wedge |m_2[k]| \le r\,) \ \wedge\ \vec{a} \ne \vec{0}\Big] \ \le\ \left(\frac{2r+1}{q}\right)^\ell + \mathsf{negl}(\lambda) \tag{1}$$

assuming [LT22, Conj 8.4] holds when replacing n there with $n-1$.

Theorem 8.7. For any $\epsilon_{\text{p}} > 0, \epsilon_{\text{n}} > 0, P = \mathsf{poly}(\lambda), G = \mathsf{poly}(\lambda)$, $\mathsf{FGOMR1}$ in Algorithm 5 (patched as above) is a DoS-resistant Fixed Group Oblivious Message Retrieval scheme, when instantiated with any PRF f and random oracle h, assuming the hardness of Ring-LWE and [LT22, Conj 8.4].

Proof sketch. The completeness, soundness, and compactness all follow in a straightforward way. We prove the DoS resistant by contradiction.

Notice that since a group only contains up to G recipients, if the adversary wants the clue $(\vec{a}, \alpha_1, \ldots, \alpha_\ell, \beta, \mathsf{s})$ to be detected as pertinent to $G+1$ honest recipients, it should be detected as pertinent by recipients in two different groups. Therefore, for two different honest recipients with auxiliary keys $\mathsf{dk}, \mathsf{dk}'$ in two different groups, let $Z \leftarrow_{\mathsf{s}} \mathbb{Z}_t^{L\times G'}$, we can generate ciphertext $(\vec{a}, \vec{b}_1 \leftarrow \vec{b} - \langle \mathsf{dk} \times Z, \alpha_1\rangle || \ldots || \langle \mathsf{dk} \times Z, \alpha_\ell\rangle, \vec{b}_2 \leftarrow \vec{b} - \langle \mathsf{dk}' \times Z, \alpha_1\rangle || \ldots || \langle \mathsf{dk}' \times Z, \alpha_\ell\rangle$. This breaks Theorem 8.6 and thus $\mathsf{FGOMR1}$ is DoS resistant. □

9 Performance Evaluation

9.1 Methodology

We implemented optimized OMR scheme $\mathsf{OMR\text{-}OPT}$ in Section 5.3, AGOMR schemes $\mathsf{AGOMR1}$, $\mathsf{AGOMR2}$, $\mathsf{AGOMR3}$ in Section 6, and our FGOMR schemes $\mathsf{FGOMR1}$, $\mathsf{FGOMR2}$ in Section 7 , in a C++ library (released as open source). Our GOMR implementations are based on our optimized OMR scheme $\mathsf{OMR\text{-}OPT}$ in Section 5.3 We used the OMR library [OMR21] as our base implementation, the PALISADE library [PAL21] for PVW encryption, and the SEAL library [Mic20] with Intel-HEXL acceleration [BKS+21] for BFV scheme. We benchmarked these schemes on a Google Compute Cloud `e2-standard-2` with 8GB RAM (except where noted otherwise) with several parameter settings.

Parameters. For fair comparison to prior works, we reuse all the parameters from [LT22, §10.1] (except for N): $\bar{k} = 50$, $\epsilon_{\text{p}} = 2^{-21}$, $\epsilon_{\text{n}} = 2^{-30}$, $\mathsf{PVW}.(n, w, \ell, q, \sigma) = (450, 16000, 4, 65537, 1.3)$ and $\mathsf{BFV}.(D, Q, t) = (2^{15}, \sim 2^{850}, 65537)$[9]; payload size is 612 bytes. We change N from 500000 to 32768, as for any $N > 32768$, the detector runtime for all the schemes (including [LT22]) scales linearly with $\lceil N/32768 \rceil$ and other metrics including key sizes, clue size, digest size, and recipient runtime remain unchanged.

Application parameters. We fix $P = 2^{60}$, $G' = G + 4$, [10] $\epsilon_{\text{DI}} \leq 2^{-128}$ (defined in Section 6.2 [11]). For $\mathsf{AGOMR3}$, $\mathsf{AGOMR2}$, $\mathsf{FGOMR1}$, $\mathsf{FGOMR2}$, we set $I = L = \max(G', 8)$, such that id and dk are unique with overwhelming probability. For $\mathsf{AGOMR1}$, we set $\mathcal{D} = \mathbb{Z}_{2^{127}-1}$ for this same reason and $2^{127} - 1$ is a prime (such that an inverse exists for any field element).

9.2 Evaluation Results

Representative costs. Table 2 summarizes the main cost metrics of all our schemes and the baseline, for the parameters above for functionality and asymptotic costs, see Fig. 2. . We have also tested $N = 2^{17}, 2^{18}, 2^{19}, 2^{20}$, and their costs are essentially the same for $N = 2^{15}$, i.e., all the numbers in Table 2 are unchanged within the margin of measurement error, except that amortized digest size decreases: the total digest size remains the same while N grows.[12]

[9] We set $\mathsf{BFV}.Q$ according to the homomorphic multiplicative depth in each scheme, for 128-bit security: $\log(Q){\approx}789$ in $\mathsf{OMRp2}$, $\log(Q){\approx}810$ in $\mathsf{OMR\text{-}OPT}, \mathsf{AGOMR4}, \mathsf{AGOMR1}, \mathsf{FGOMR1}, \mathsf{FGOMR2}$, $\log(Q){\approx}838$ in $\mathsf{AGOMR2}$, and $\log(Q){\approx}868$ in $\mathsf{AGOMR3}$ [APS15].

[10] This G' guarantees that for any G, we have $\epsilon_{\text{DS}} \approx 2^{-48}$ (defined in Section 6.2 which is the probability that the matrix calculated by multiplying a full-rank matrix and a random matrix is not full rank). In the case that all the extended id's (or dk's) are linearly independent, the probability that our GOMR schemes fail to send a message is $\epsilon_{\text{DS}} \approx 2^{-48}$. However, this can be fixed by performing reject sampling (i.e., resample a Z until a full rank Z is found), at the cost of a minor leakage: given Z, there might exist some combination of recipients that the message cannot be sent.

[11] ϵ_{DI} is the probability for there to exist any combination of G id's such that their extended version forms a matrix with rank $< G$.

[12] This runtime gives us that the cost per million message scanned is ∼\$3.4, given that our instance costs $\sim 1.89 \cdot 10^{-5}$/sec and that it costs ∼1810 seconds per million messages.

	Ad-hoc GOMR					Fixed GOMR	
	OMRp2 §6.1 + [LT22]	OMR-OPT §6.1 +Thm 5.1	AGOMR1 Theorem 6.1	AGOMR2 Theorem 6.2	AGOMR3 Thm 6.4	FGOMR1 Thm 7.3	FGOMR2 Theorem 7.4
Detector preprocessing time (sec/msg)	N/A	N/A	N/A	0.0066	0.0232	0.0041	N/A
Detector computation time (sec/msg)	5.205	1.844	0.117	0.168	0.181	0.123	0.119
Sender computation time (sec/msg)	0.131	0.131	0.136	0.132	0.136	0.020	0.020
Clue size (bytes/msg)	11472	11472	11472	15312	15312	1108	1092
Clue key size (bytes)	133K per recipient	133K per recipient	133K per recipient	133K per recipient	133K per recipient	1.56M per group	1.56M per group
Detection key size (bytes)	137M	139M	139M	143M	143M	142M	139M
Digest size (bytes/msg)	35						
Recipient time (sec)	0.026						

Table 2: Comparison of cost metrics, given $N = 2^{15}, k = \bar{k} = 50, G = 15, G' = 19, I = L = 81$. Costs are per message, per recipient, except for the preprocessing time, which can be shared by ≥ 1 recipients. For the functionality differences, see Fig. 2 for details.

We see that for all our GOMR schemes, the detector online time is at least an order of magnitude faster than the baseline scheme OMRp2[LT22, §7.4] + §6.1. FGOMR schemes have better sender time and clue size, though the clue key size is larger (since it represents the entire group rather than a single recipient). AGOMR has a comparable sender time and clue size to the baseline. The detection key size, digest size, and recipient runtime are all relatively similar across all schemes.

Furthermore, our OMR-OPT shows an improvement of ~ 2.5x faster detector time standalone as an OMR scheme.

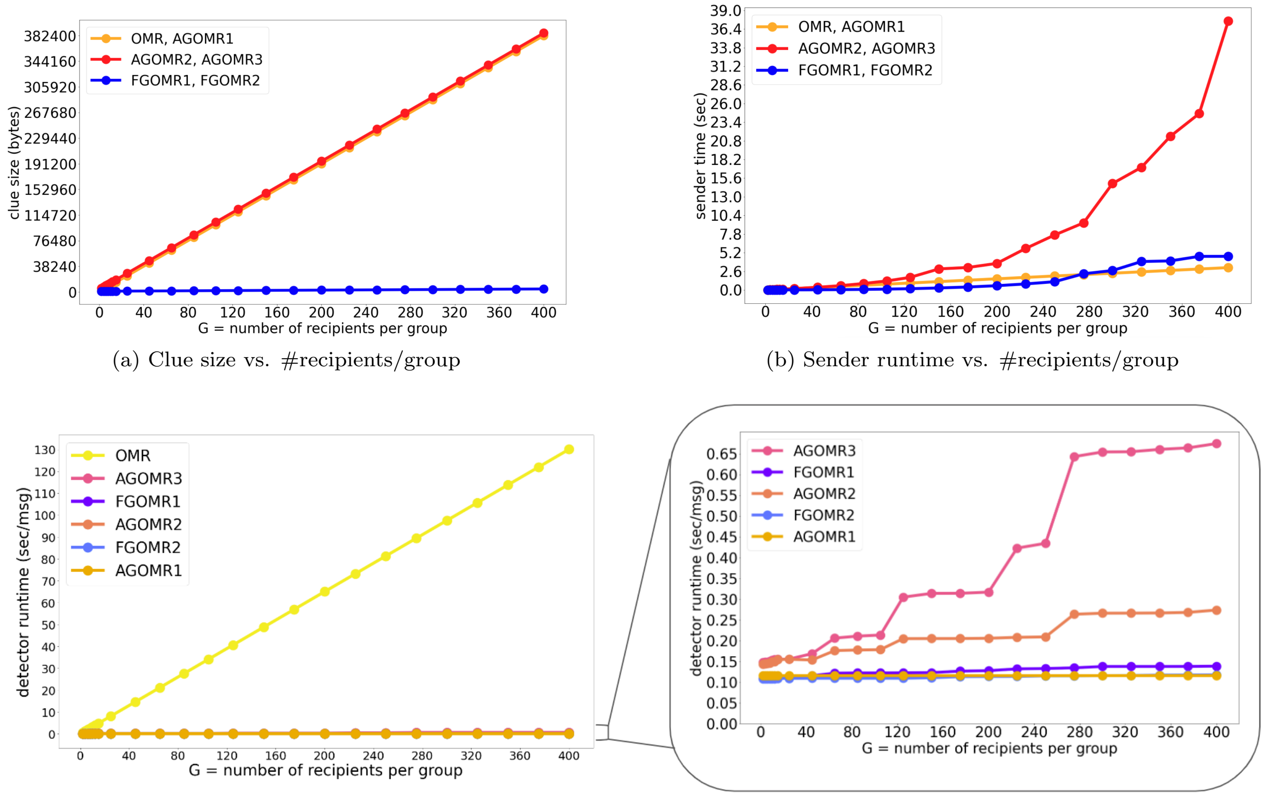
Costs vs. group size G. Fig. 10 shows how the log of detector runtime, the log of sender time, and the log of clue size scale with the log of the number of recipients per group. (Other metrics are independent of G, and thus given by Table 2.) OMR-OPT has roughly the same behavior as OMRp2, so we leave it out of the comparison for better visualization. For readability, we merge the schemes that have very minor differences ($< 5\%$).

We benchmarked group sizes $\{2, 4, 6, 8, 10, 12, 15, 25, 45, 65, 85, 105, 125, 150, 175, 200, 225, 250, 275, 300, 325, 350, 375, 400\}$. The corresponding id sizes for AGOMR, or the dk size for FGOMR, are $\{19, 28, 38, 47, 57, 66, 81, 128, 223, 318, 413, 508, 603, 722, 841, 959, 1078, 1197, 1316, 1434, 1553, 1672, 1791, 1909\}$. Due to memory constraints (simply because every clue is already too large to store in memory for a large group size), AGOMR2, and AGOMR3 with group size ≥ 45 use a GCP `e8-highmem-64` instance, 64GB RAM (with a 128GB balanced disk). Other schemes and other group sizes still use the `e2-standard-2` instance type, 8GB RAM. Note that the runtime of the instance `e8-highmem-64` is roughly the same as `e2-standard-2` (about 2.5% faster) for the same schemes with the same group sizes, and thus it is still very continuous in the plot and hard to spot the difference.

As BFV uses power-of-two cyclotomic rings, the underlying ring dimension is a power-of-two. Thus, when doing matrix multiplication between matrices of sizes $(a \times b)$ and $(b \times c)$, for better efficiency, we first round up a, b, c to the nearest power of two. The round-ups in our schemes cause the jumps in Fig. 10.

All schemes have two to three orders of magnitude faster detector runtime compared to the baseline. The detector time of our main schemes AGOMR3, AGOMR2 increases most rapidly among all, because of the large amount of matrix multiplications they need to perform as shown in Fig. 2. Note that the runtime of our GOMR schemes are calculated as $T_p/10 + T_o$ where T_p is the detector preprocessing time, and T_o is the detector online computation time listed in Table 2. In other words, the runtime is computed based on ten recipients sharing the preprocessing time. For most of our schemes and parameters, the preprocessing will only take $< 5\%$ of the total time. Thus, when the preprocessing time is not amortized over multiple recipients (as in Table 2), in the worst case (AGOMR, $G = 400$), the total runtime of the detector will be only less than two times larger. On the other hand, to further reduce the amortized cost, preprocessing can of course be shared by more recipients.

The clue size for our FGOMR scheme is smaller and grows more gently compared to AGOMR and the



(a) Clue size vs. #recipients/group

(b) Sender runtime vs. #recipients/group

(c) Detector runtime vs. #recipients/group

Figure 10: Figures show how the log of clue size, the log of sender time, and the log of detector time scale with G the number of recipients in a group. Other parameters are as in Table 2.

baseline. The sender time, in our schemes, grows super-linearly with the group size (due to solving a linear system of equations), compared to linear growth for the baseline; but is still less than a second.

10 Applications

10.1 Secure Group Messaging

The most direct motivation for GOMR is secure group messaging. As discussed in Section 1, Ad-hoc GOMR suits groups of arbitrary recipients chosen on the fly, whereas Fixed GOMR suits applications with well-defined groups such as group chats and mailing lists.

Concrete costs. We use the same parameters as benchmarked in Section 9 (in the absence of detailed data about deployed systems). For a GCP `e2-standard-2` instance (with a 32GB balanced disk), the average cost is $\sim\$1.89 \cdot 10^{-5}$/sec.[13] Thus, for 32768 messages, group size $G = 15$, our GOMR schemes cost range from \$0.07 to \$0.11, while if the baseline OMRp2 from [LT22, §7.4] costs ~\$3.2.

Varying group size. Group size varies between messages. The group size parameter G can be adaptively changed between messages (at a cost of leaking the group size), or groups can be padded to some standard sizes or randomized sizes (to minimize information leakage but (at a cost in efficiency). This tradeoff is information-theoretically inherent.

[13] Estimate using `https://cloud.google.com/products/calculator`. Communication cost is negligible: $< \$10^{-9}$/msg egress.

10.2 Privacy-Preserving Blockchains

Transaction Batching. In many blockchain protocols, it is cost-effective to aggregate multiple transactions into a single one with multiple recipients. This is a natural application for AGOMR, especially with the multi-payload extension of Section 6.3 that allows each recipient to retrieve just the pertinent portion.

Private Bitcoin. Concretely, in Bitcoin, transactions with 2 recipients consist of $\sim$74% of all transactions, and transactions with more than 2 recipients take $\sim$10% of all transactions. If Bitcoin became privacy-preserving, using our AGOMR scheme $\mathsf{AGOMR3}$ instead of directly using OMR scheme instantiated with our optimized OMR scheme $\mathsf{OMR\text{-}OPT}$ results in a $\sim$1.5x speedup overall ($\sim$4x speedup if instantiated with $\mathsf{OMRp2}$ in [LT22, §7.4]), at the cost of clue size grown by $\sim$3x. Note that in the estimation, we do not hide the group size (i.e., group size of each transaction is still public and vary by transactions).

Blockchain Auditing. In privacy-preserving cryptocurrencies (like Zcash or Monero), a user may want to share the transaction information with service providers such accountants or auditors, or with pertinent authorities. Currently, privacy-preserving cryptocurrencies use a viewing key to allow other users to view the transactions. This entails extra detection costs: every viewing-key holder needs to separately look for transactions in every account they can view. For example, using OMR, an accountant would need to perform a separate OMR retrieval for each of their hundreds of clients.

Each such group is represented by a single FGOMR group clue key, and senders are instructed (at the protocol level) to use this clue key when transacting with this account. Subsequently, each recipient performs retrievals only for their own key, yet will receive all messages addressed to all the groups (e.g., accounting clients) of which they are members.

A limitation of this approach is that to grant or revoke view keys, the group clue key needs to be regenerated and republished.

Acknowledgements

We are grateful to Xifan Yu for discussing the probability bound on id length for maliciously crafted groups in Section 6. We also thank the valuable suggestions from the anonymous reviewers.

This material is based upon work supported by Algorand Centres of Excellence programme managed by Algorand Foundation; DARPA under Contract No. HR001120C0085; the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research under award number DE-SC-0001234; JPMorgan Chase & Co; LexisNexis Risk Solutions; and Mastercard Center for Inclusive Growth. 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 Algorand Foundation, the United States Government, DARPA, DOE, JPMorgan Chase & Co. or its affiliates, or other sponsors.

References

[ACDT20] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 248–277, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.

[ACDT21] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Modular design of secure group messaging protocols and the security of mls. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, page 1463–1483, New York, NY, USA, 2021. Association for Computing Machinery.

[AHK+22] Joël Alwen, Dominik Hartmann, Eike Kiltz, Marta Mularczyk, and Peter Schwabe. Post-quantum multi-recipient public key encryption. Cryptology ePrint Archive, Paper 2022/1046, 2022. CCS 2023. URL: `https://eprint.iacr.org/2022/1046`.

[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.

[BBKS07] Mihir Bellare, Alexandra Boldyreva, Kaoru Kurosawa, and Jessica Staddon. Multirecipient encryption schemes: How to save on bandwidth and computation without sacrificing security. *IEEE Transactions on Information Theory*, 53(11):3927–3943, 2007. `doi:10.1109/TIT.2007.907471`.

[BBS02] Mihir Bellare, Alexandra Boldyreva, and Jessica Staddon. Randomness re-use in multi-recipient encryption schemeas. In Yvo G. Desmedt, editor, *Public Key Cryptography — PKC 2003*. Springer Berlin Heidelberg, 2002.

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

[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*, 2017.

[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.

[BLMG] Gabrielle Beck, Julia Len, Ian Miers, and Matthew Green. Fuzzy message detection. ACM CCS 2021.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *CRYPTO 2012*, LNCS. Springer, 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.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th FOCS*, pages 41–50, Milwaukee, Wisconsin, October 23–25, 1995. IEEE Computer Society Press. `doi:10.1109/SFCS.1995.492461`.

[CGN98] Benny Chor, Niv Gilboa, and Moni Naor. Private information retrieval by keywords, 1998. URL: `http://eprint.iacr.org/1998/003`.

[Con23] Deirdre Connolly. Oblivious Message Retrieval. `https://zfnd.org/oblivious-message-retrieval/`, March 2023. Zcash foundation.

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

[FLS22] Nils Fleischhacker, Kasper Green Larsen, and Mark Simkin. How to compress encrypted data. Cryptology ePrint Archive, Paper 2022/1413, 2022. URL: `https://eprint.iacr.org/2022/1413`.

[FN94] Amos Fiat and Moni Naor. Broadcast encryption. In Douglas R. Stinson, editor, *Advances in Cryptology — CRYPTO' 93*, pages 480–491, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[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, Paper 2012/144, 2012.

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

[GOM23] Group oblivious message retrieval implementation. `https://github.com/ObliviousMessageRetrieval/ObliviousMessageRetrival`, May 2023.

[HBHW] 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`.

[HKP+21] Keitaro Hashimoto, Shuichi Katsumata, Eamonn Postlethwaite, Thomas Prest, and Bas Westerbaan. A concrete treatment of efficient continuous group key agreement via multi-recipient PKEs. pages 1441–1462. ACM Press, 2021. `doi:10.1145/3460120.3484817`.

[HKP22] Keitaro Hashimoto, Shuichi Katsumata, and Thomas Prest. How to hide metadata in mls-like secure group messaging: Simple, modular, and post-quantum. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, page 1399–1412, New York, NY, USA, 2022. Association for Computing Machinery.

[Jan22] Janmajayamall. Oblivious message retrieval [in Aztec]. `https://hackmd.io/@GmJnylgPT2mD6YGCz2uVkA/Hk4bt1eBj`, December 2022.

[JLM23] Sashidhar Jakkamsetti, Zeyu Liu, and Varun Madathil. Scalable private signaling. Cryptology ePrint Archive, Paper 2023/572, 2023.

[KKPP20] Shuichi Katsumata, Kris Kwiatkowski, Federico Pintore, and Thomas Prest. Scalable ciphertext compression techniques for post-quantum KEMs and their applications. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 289–320, Daejeon, South Korea, December 7–11, 2020. Springer, Heidelberg, Germany. `doi:10.1007/978-3-030-64837-4_10`.

[Kur] Kaoru Kurosawa. Multi-recipient public-key encryption with shortened ciphertext. In David Naccache and Pascal Paillier, editors, *Public Key Cryptography*. Springer Berlin Heidelberg.

[Lew21] Sarah Jamie Lewis. Discreet log #1: Anonymity, bandwidth and Fuzzytags, Feb 2021. URL: `https://openprivacy.ca/discreet-log/01-anonymity-bandwidth-and-fuzzytags/`.

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

[LT22] Zeyu Liu and Eran Tromer. Oblivious message retrieval. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022*, pages 753–783, Cham, 2022. Springer Nature Switzerland. Full version: Cryptology ePrint Archive 2021; internal citations follow the latter's numbering.

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

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

[MSS+22] Varun Madathil, Alessandra Scafuro, István András Seres, Omer Shlomovits, and Denis Varlakov. Private signaling. USENIX Security 2022, 2022.

[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.

[PPS14] Alexandre Pinto, Bertram Poettering, and Jacob C.N. Schuldt. Multi-recipient encryption, revisited. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '14, New York, NY, USA, 2014. Association for Computing Machinery.

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

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

[SGGC14a] Daniel Salmond, Alex Grant, Ian Grivell, and Terence Chan. On the rank of random matrices over finite fields. *arXiv preprint arXiv:1404.3250*, 2014.

[SGGC14b] Daniel Salmond, Alex J. Grant, Ian Grivell, and Terence Chan. On the rank of random matrices over finite fields. *CoRR*, 2014. `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, Paper 2021/1180, 2021.

[SPB22] István András Seres, Balázs Pejó, and Péter Burcsi. The effect of false positives: Why fuzzy message detection leads to fuzzy privacy guarantees? In Ittay Eyal and Juan Garay, editors, *Financial Cryptography and Data Security*, pages 123–148, Cham, 2022. Springer International Publishing.

[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.

Adversary		**Challenger**
	$\xleftarrow{\quad \mathsf{pp} \quad}$	$\mathsf{pp} \leftarrow \mathsf{GenParams}(1^\lambda, \epsilon_{\mathrm{p}}, \epsilon_{\mathrm{n}}, G, P)$
		$(\mathsf{sk}_j, \cdot) \leftarrow \mathsf{PersonalKeyGen}(\mathsf{pp})$ for $j \in [P]$
Query Phase I: for $i = 1, \cdots, N$		
Choose recipient set $Y_i \subseteq [P]$, where $\lvert Y_i \rvert \leq G$	$\xrightarrow{\quad Y_i \quad}$	
	$\xleftarrow{(\mathsf{gPKshare}_{j,i})_{j \in Y_i}}$	$\mathsf{gPKshare}_{j,i} \leftarrow \mathsf{GroupKeyGenAux}(\mathsf{pp}, \mathsf{sk}_j, Y_i)$, for $j \in Y_i$
Challenge Phase: Choose $i_0 \neq i_1 \in [N]$, let $Y' = Y_{i_0} \cap Y_{i_1}$		
Choose $Z \subseteq Y'$	$\xrightarrow{\quad Z \quad}$	
$(\mathsf{sk}'_j, \cdot) \leftarrow \mathsf{PersonalKeyGen}(\mathsf{pp}; r_j)$ for $j \in Z$	$\xleftarrow{(\mathsf{sk}_j)_{j \in Z}}$	
For $i \in [i_0, i_1]$: $\mathsf{gPKshare}'_{j,i} \leftarrow \mathsf{GroupKeyGen}(\mathsf{pp}, \mathsf{sk}'_j, Y_i), j \in Z$ Choose $Z_i \subseteq Y_i$, and $(\mathsf{gPKshare}'_{j,i})_{j \in Z_i}$ Let $\mathsf{gPKshare}'_{j,i} = \mathsf{gPKshare}_{j,i}, j \in Y_i \setminus (Z_i \cup Z)$		
Choose a payload x	$\xrightarrow[((\mathsf{gPKshare}'_{j,i})_{j \in Y_i})_{i \in [i_0, i_1]}]{i_0, i_1, x, Z_0, Z_1}$	$b \leftarrow\$ \{0, 1\}$
		$\mathsf{pk}_{\mathsf{clue}b} \leftarrow \mathsf{GroupKeyGen}(\mathsf{pp}, (\mathsf{gPKshare}'_{j,i_b})_{j \in Y_{i_b}}, Y_{i_b})$
	$\xleftarrow{\quad c \quad}$	$c \leftarrow \mathsf{GenClue}(\mathsf{pp}, \mathsf{pk}_{\mathsf{clue}b}, Y_{i_b} \setminus Z_{i_b}, x)$
Query Phase II: same as **Query Phase I**.	$\xrightarrow{\quad b' \quad}$	(Adversary wins if $b = b'$)

Figure 11: Computational Privacy game for Fixed Group OMR with semi-maliciously chosen secret keys

A Strengthed Privacy Definition of FGOMR

We give the strengthed privacy definition of FGOMR in Fig. 11 as discussed in Theorem 4.4.

B Sparse Random Linear Coding (SRLC)

SRLC, defined in [LT22, §6.3.1], is used to compress all the payloads of the pertinent messages into the digest for the OMR and GOMR constructions introduced in [LT22] and this paper. It essentially creates weighted linear combinations of those payloads. Moreover, the combinations are sparse: each message is assigned to just a few combinations (i.e., most of the weights are chosen as zeros), which is important for efficient homomorphic computation of these combinations on ciphertexts. The combinations are included in the digest, such that the recipients can later decode them and obtain the payloads.

A formal definition is provided as in Definition B.1 (adapted from [LT22, §6.1.3]). SRLC is only used in the compression step in Section 5.1 on the detector side in all GOMR schemes, and the usage remains unchanged from [LT22, §7.4] for all our GOMR constructions.

Two different SRLC algorithms $\mathsf{SRLC1}$ [LT22, Alg 3] and $\mathsf{SRLC2}$ [LT22, Alg 4] are introduced in OMR [LT22, §6.3.1], where $\mathsf{SRLC1}$ has clean analysis using known bounds, and suffices for our asymptotic results and analysis for Theorem 6.4 and Theorem 7.3. $\mathsf{SRLC2}$ is simpler, faster, and smaller, but relies on empirical estimation for completeness and we thus use it for our evaluation in Section 9.

Definition B.1. 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):

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