A Performance Evaluation of Pairing-Based Broadcast Encryption Systems*

Arush Chhatrapati
Henry M. Gunn High School
Palo Alto, CA, USA
arush.chhatrapati@gmail.com

Susan Hohenberger
Johns Hopkins University
Baltimore, MD, USA
susan@cs.jhu.edu

James Trombo
George Mason High School
Falls Church, VA, USA
jamestrombo@gmail.com

Satyanarayana Vusirikala
University of Texas at Austin
Austin, TX, USA
satya@cs.utexas.edu

November 16, 2021

Abstract

In a broadcast encryption system, a sender can encrypt a message for any subset of users who are listening on a broadcast channel. The goal of broadcast encryption is to leverage the broadcasting structure to achieve better efficiency than individually encrypting to each user; in particular, reducing the bandwidth (i.e., ciphertext size) required to transmit securely, although other factors such as public and private key size and the time to execute setup, encryption and decryption are also important.

In this work, we conduct a detailed performance evaluation of eleven public-key, pairing-based broadcast encryption schemes offering different features and security guarantees, including public-key, identity-based, traitor-tracing, private linear and augmented systems. We implemented each system using the MCL Java pairings library, reworking some of the constructions to achieve better efficiency. We tested their performance on a variety of parameter choices, resulting in hundreds of data points to compare, with some interesting results from the classic Boneh-Gentry-Waters scheme (CRYPTO 2005) to Zhandry's recent generalized scheme (CRYPTO 2020), and more. We combine this performance data and knowledge of the systems' features with data we collected on practical usage scenarios to determine which schemes are likely to perform best for certain applications, such as video streaming services, online gaming, live sports betting and smartphone streaming. This work can inform both practitioners and future cryptographic designs in this area.

*This is the full version of a paper that will be presented at ACNS 2022 (20th International Conference on Applied Cryptography and Network Security).

Contents

1 Introduction

In a broadcast encryption system [FN93], a sender can encrypt a message for any subset of users who are listening on a broadcast channel. We focus on public-key systems, where there is a public system key that allows anyone to encrypt a message to any set S of his choice out of an established set of N users. The public system key is established by a master authority, who also distributes individualized secret keys to each user in the system. If a user is in the set S for a particular broadcast, then she can decrypt that broadcast using her secret key. A critical security property for these systems is collusion resistance, which guarantees that users not in S learn nothing about the broadcast message. Some schemes offer a traitor-tracing functionality that protects against digital piracy; specifically, it guarantees that if one or more malicious users work together to release piracy information (e.g., software or a key) that decrypts on the broadcast channel, then this piracy information can be traced back to at least one of them.

The goal of broadcast encryption is efficiency. In particular, the goal is to leverage the broadcasting structure to achieve better efficiency than individually encrypting to each user. This can result in huge practical savings. To measure the concrete performance benefits offered by various broadcast encryption systems, for several different sizes of system users N and encryption subsets $S \subset N$, we will compare each broadcast encryption scheme in terms of bandwidth (i.e., ciphertext size), public and private key size, and setup, key generation, encryption and decryption times. We focus on pairing-based broadcast systems, since this is the most promising algebraic setting for reducing bandwidth and obtain fast runtimes. We also compare the broadcast schemes to an optimized "baseline" scheme[1] derived from ElGamal encryption [Gam85] with shared parameters (see Section 3.2) that individually encrypts to each user in the broadcast set S.

Our Contributions and Results. To the best of our knowledge, this work is the only current detailed performance study of public-key, pairing-based broadcast encryption systems. Although schemes can be loosely grouped and compared at the asymptotic level for performance purposes, the tradeoffs, underlying constant factors, scalability and differing system features could significantly impact various applications. To provide the community with a solid foundation for comparisons, this work includes the following:

- We collected the most promising 11 public-key, pairing-based broadcast encryption systems, which are detailed in Table 1 of Section 3. In some cases, we made efficiency-focused alterations to the schemes, such as creating a separate setup and key generation function or finding the most efficient asymmetric pairing implementation for a scheme presented symmetrically. Any change from the original publication is documented in Section 3 and all changes were made to optimize the performance.

- Section 2 defines the different features and security guarantees offered by these schemes, including identity-based, traitor-tracing, private linear, augmented systems and more.

- We implemented the 11 broadcast systems using the MCL pairings library (currently employed by some cutting-edge cryptocurrency companies) and the baseline ElGamal system using OpenSSL. We ran hundreds of tests on these systems for various parameter choices, reporting on those results in Section 3. This is a contribution in terms of providing the community with data and public reference implementations; additionally careful implementation is also important for rooting out any potential issues in prior publications. In the course of our study, we discovered a technical issue with Construction 7 of Zhandry [Zha20] (CRYPTO 2020). We communicated a potential fix, which Zhandry accepted and graciously acknowledged in Section 1.4 of his updated work [Zha20]. Thus, this implementation effort has also been useful as an additional verification process for prior work.

[1] Because the Section 3.2 baseline scheme will not require the pairing operation, it is implemented using an optimized elliptic curve group, whose elements are even smaller; thus requiring real performance gains from the broadcast systems to overtake it.

- In Tables 8 and 9, we document that individual encryption is more efficient than broadcast encryption for systems with 100 users or less. The $100 < N < 10,000$ range is a gray area where there are tradeoffs to be made. But once a system's users exceed 10,000, broadcast encryption dominates the individual encryption (baseline) in overall performance.

- To understand which broadcast system offers the "best" performance, we researched the yearly reports and shareholder letters of companies such as Nvidia [Nvi20], Disney [Com19] and Netflix [Cor20] to understand the current performance demands of some interesting applications for broadcast encryption. We summarize our findings in Section 4. We start with the classic application of video streaming and then explore the emerging applications of online gaming, live sports betting and peer-to-many applications. In a nutshell, if traitor tracing is not required, we found that the classic Boneh-Gentry-Waters system [BGW05, $\mathcal{S}3.2$] provides the best tradeoffs for video streaming and online gaming. Zhandry's generalized nonrisky system (see Section 3.8.10) can be tuned to optimize a parameter of interest (e.g., ciphertext size), although this usually results in another parameter (e.g., decryption time) becoming infeasible. For live sports betting, the smaller number of users and the importance of encryption speed make Gentry-Waters [GW09] the preferred choice. We found the private tracing system of Gentry, Kumarasubramanian, Sahai and Waters [GKSW09] to provide the best overall system performance when tracing is needed, but it may not be fast enough for live streaming applications. For peer-to-many applications, we favored Boneh-Gentry-Waters [BGW05, $\mathcal{S}3.1$] when many keys must be generated. Finally, we discovered that none of the identity-based broadcast systems (IBBE) were practical for large user applications, so a practical IBBE remains an interesting open research problem.

We believe this timely implementation study will inform practitioners as they look to harness the performance savings of broadcast encryption, while also providing context for future broadcast encryption designs.

On CPA and CCA Security for Broadcast Systems. It is important to remark that nearly all of the schemes we implemented, including the ElGamal baseline, achieve security against chosen plaintext attacks [GM84] (CPA), while NIST recommends that deployed systems achieve a stronger notion of security against chosen ciphertext attacks [NY90, RS91, DDN00] (CCA). While efficient general transformations from CPA to CCA exist for public key encryption [FO99], it is not clear if these can be applied to broadcast encryption systems without compromising some of their functionality. Important open research questions in the area of broadcast encryption include a strengthening of existing definitions (including extensions such as traitor tracing, augmented, etc.) to realize a suitable notion of CCA security and providing an efficient generic transformation from CPA to CCA for these systems. For now, CCA transformations must be done individually and manually, so this work can help narrow the focus to the most suitable candidate.

1.1 Applications of Broadcast Encryption

Online Video Streaming The most commonly referenced use case for broadcast encryption is online video streaming services like Netflix, Hulu, Quibi, etc. Users with permission are given access to a myriad of different videos, and ideally bandwidth usage and client side decryption processing requirements need to be minimized so that users can watch the videos in real time and can watch those videos on any device, regardless of processing capability. The user numbers for these services are vast. During the second quarter of 2020 [Cor20], Netflix and Hulu had at least 190 million and 30 million users respectively. For these media streaming cases, we would recommend using the classic Boneh-Gentry-Waters [BGW05] scheme as it provides the best combination of short ciphertexts and fast decryption times even for large user sets. For $N = 1$ million users, the [BGW05, $\mathcal{S}3.1$] variant provides the best bandwidth at 96B per ciphertext while decryption takes 350ms and the [BGW05, $\mathcal{S}3.2$] variant provides the fastest decryption at 1.6ms with a 32KB ciphertext. Either of these are reasonable choices, although if the bandwidth isn't a problem, we'd recommend [BGW05, $\mathcal{S}3.2$] due to its smaller public key size (of 160KB, whereas [BGW05, $\mathcal{S}3.1$] requires 128MB for 1 million users). For very large user datasets (e.g., in the 190 million range), using [BGW05, $\mathcal{S}3.2$] instead of [BGW05, $\mathcal{S}3.1$] becomes even more important, as the former's public keys scale with $\sqrt{N}$ while

the latter scale with N. Both [BGW05] schemes were proven secure in the static attacker model; if one wants the stronger adaptive attacker security, Waters [Wat09] offers this and small 861B ciphertexts, although the public key sizes grow to 32MB for $N = 1$ million. Both of the identity-based systems [GW09, GW19] require *hours* to decrypt a single ciphertext when N is 1 million, so they are not contenders.

The performance hit from [GKSW09] (the best performing traitor tracing scheme) vs. [BGW05] could be worth it for the chance to combat revenue reducers like piracy. For $N = 1$ million users, the decryption time of [GKSW09] doubles (over [BGW05, $\mathcal{S}$3.2]) to 3.2ms while the ciphertext size grows by a factor of 19 to 605KB – larger, but reasonable on fast networks. The public key size roughly triples to 477KB. Zhandry's risky traitor tracing scheme [Zha20, $\mathcal{S}$9.3] provides the best bandwidth for tracing schemes at only 384B, but the decryption time explodes to an infeasible 19 hours (for $N = 1$ million). Zhandry's nonrisky, post-user expansion compiler version of his scheme (see Section 3.8.10) has decryption times that are comparable to those for [GKSW09] and [GKSW10] for $N = 1$ million, but the encryption time balloons to over 1 minute and the ciphertext size jumps from roughly 600KB to almost 4MB (observe Table 16 when $a = 2/3$.) One potential benefit is that the public/private key sizes of Zhandry's Section 3.8.10 scheme are smaller, but that likely won't offset the additional encryption and bandwidth overhead.

Constraint Summary: The scheme needs to scale to 1 million users or more, with a small bandwidth overhead and fast (client side) decryption times. Encryption times are less of a concern, but traitor tracing may be needed.

Recommendation: Use [BGW05, $\mathcal{S}$3.2] (for fastest decryption and scalable public key size). See Section 3.3. If traitor tracing is required, use [GKSW09]. See Section 3.8.

Online Game Streaming Online game streaming is another form of media streaming that is becoming increasingly prevalent. Users receive high quality (resolution and frames per second) game data and give the server data like their keystrokes and mouse clicks in game. This system allows users to play games that have a performance requirement beyond what their client side device is capable of. In comparison to video streaming, online game streaming has a more stringent data speed requirement, as any wasted time could result in a subpar player experience. The most popular service, Nvidia GeForce Now, has around a million users [Nvi20], but it is a growing industry and the ceiling for game streaming services could be having user numbers on par with video streaming services. For this case, we would recommend [BGW05, $\mathcal{S}$3.2] or [Wat09]. Both schemes have ciphertexts under 1KB even for 1 million users, but the primary cost for both is a jump in public key size of 128MB and 32MB respectively. The scheme described in [Wat09] offers stronger provable security, while [BGW05, $\mathcal{S}$3.2] offers decryption times that are two orders of magnitude faster.

While the live traitor tracing functionality could be useful, the difference in performance could make a notable difference for users. Perhaps [GKSW09] could be used in situations where most of the game data is preloaded on the client side, and the live data is sent out live and unencrypted or from a faster performing scheme like [BGW05]. This could be a hybrid combination, allowing usage of the traitor tracing functionality, and ensuring fast enough performance. We note that the baseline ElGamal scheme takes almost 3 seconds to encrypt the payload for 1 million users, which likely rules this out for live gaming applications, highlighting the power of broadcast encryption for this setting.

Constraint Summary: The system needs to scale to 1 million users or more, with a combination of bandwidth overhead and (client side) decryption times that support live interactions. Overall, it needs to balance the benefits of traitor tracing with impact on user experience.

Recommendation: Use [BGW05, $\mathcal{S}$3.2] (for fastest decryption). If faster transmission is needed for the live gaming experience, use [BGW05, $\mathcal{S}$3.1] (shortest ciphertext, but largest PK) or [Wat09] (short ciphertext and more tolerable PK size) to cut the bandwidth overhead. See Tables 8 and 9. The overhead required for traitor tracing may frustrate the live gaming experience, but if it is needed, a hybrid approach using [GKSW09] for the most sensitive data may work. See Tables 15 and 14.

Live Sports Betting A novel use case for broadcast encryption arrives with the emergence of live sports betting. Due to the new developments in wireless data speeds with the emergence of 5G technologies, some

major companies are developing capabilities for in-person spectators to make bets on their mobile phones throughout a game, utilizing continually updating betting lines given the events happening within the game. Broadcast encryption could be used to quickly send out information to users about how much current bets are worth to cash out and the current betting lines, all in realtime. In this use case, the total speed is the most important factor (making the encryption time more relevant here), and the total number of users is within a pretty regular range (N = 30,000 to 70,000), which is much smaller than the user amounts in some other use cases. Like with online game streaming, broadcast encryption offers real performance savings over individually encrypting with ElGamal; when $N = 100,000$ the encryption plus decryption time of ElGamal is roughly 10 times that of [GW09, $\mathcal{S}$3.1] or [BGW05, $\mathcal{S}$3.1]. For the $N \leq 100,000$ range, the public keys of [BGW05, $\mathcal{S}$3.1] are 13MB, while the public keys of [GW09, $\mathcal{S}$3.1] are a more tolerable 3MB. Systems [GW09, $\mathcal{S}$3.1] and [BGW05, $\mathcal{S}$3.1] tie for the shortest ciphertexts at 96B. The fastest encryption plus decryption time is [GW09] for this user level (and this holds over a range of sizes of allowed decrypter sets S from 10% to 50% of N), although the difference (a few milliseonds) isn't likely to be observable by a human.

Constraint Summary: We are looking for a sweet spot in the 10,000 to 100,000 user range, with a combination of bandwidth overhead and (server side) encryption and (client side) decryption times that support real-time interactions.

Recommendation: Use [GW09, $\mathcal{S}$3.1] (for best bandwidth, best sum of encryption and decryption time, and public key size tolerable for $N \leq 100,000$). See Section 3.4. The overhead required for traitor tracing may frustrate the live betting experience, but if it is needed, a hybrid approach using both [GKSW09] and [GW09] may work. See Section 3.8.

Distributor Limited Applications In the above applications, we assume that the distributor (e.g., Netflix, Disney) has large computing resources at its disposal. However, we also anticipate use cases where distributor performance becomes a bottle-neck (e.g., where a person is streaming video from their smartphone to a group). Possibly in the case of a direct peer-to-many-peers type of communication, the performance of the distributor system becomes relevant, thus making the times for the Setup, KeyGen and Encrypt functions more critical. In this situation, a simple recommendation is harder to make. [BGW05, $\mathcal{S}$3.2] for example, performs the best in the case of online video streaming, but if one person was streaming video from their smartphone directly to many peers, the encryption performance of [BGW05, $\mathcal{S}$3.2] is much worse than [BGW05, $\mathcal{S}$3.1] and [GW09, $\mathcal{S}$3.1]. Due to that constraint, if there are limited resources for the distributor, using [BGW05, $\mathcal{S}$3.2] isn't a good choice. If peers are less than 100K, in a situation with a distributor bottleneck we'd recommend either [BGW05, $\mathcal{S}$3.1] or [GW09, $\mathcal{S}$3.1]. The latter has much better performance in terms of encryption times, but the prior is orders of magnitude faster during KeyGen. In a situation where many keys are regularly generated, [BGW05, $\mathcal{S}$3.1] would be preferable, but in cases where keys are generated less often [GW09, $\mathcal{S}$3.1] will have the best performance, allowing the fastest encryption.

The same consideration can be made for the traitor tracing schemes. When the amount of users is around 1K, [GKSW10] slightly outperforms [GKSW09] in both setup times and encryption times, with roughly the same decryption times and notably worse KeyGen times. In a situation with distributor performance restraints, where many keys are regularly generated, [GKSW09] will perform better. In a situation where keys are generated less often and there are less than 1K users, [GKSW10] can perform better. However, both of these schemes are outperformed by the baseline (individual encryption to each peer) until about about the 10K user level.

Constraint Summary: For the 1K to 100K user range, we are looking for a scheme that optimizes the distributor functions without sacrificing much bandwidth or client performance.

Recommendation: Use [BGW05, $\mathcal{S}$3.1] (if need to generate keys often) or [GW09, $\mathcal{S}$3.1] (otherwise). See Tables 8 and 9. If traitor tracing is required for under 10K users, the baseline (individual encryption) will likely outperform any of the tracing broadcast schemes. If traitor tracing is required for over 10K users, use [GKSW09]. Compare Tables 8 and 9 to Tables 15 and 14.

1.2 Related Work

Our study focuses on the implementation of pairings-based broadcast encryption systems. We now discuss prior implementation efforts and interesting schemes not included in this study.

The foundations for broadcast encryption were originally laid by [FN93]. An early survey on broadcast encryption [Hor03] compiled information from simple broadcast encryption schemes and traitor tracing schemes, with a special focus on broadcast encryption schemes in the subset-cover model [NNL02] .

Later, Garg, Kumarasubramanian, Sahai and Waters [GKSW09] implemented their private-linear broadcast encryption (PLBE) system. They used the (older) pairings-based crypto (PBC) library to implement the system in both the symmetric and asymmetric pairing setting. In this paper, we implement the same PLBE system in the asymmetric pairing setting in Section 3.8.5. Our runtimes are much faster than those in [GKSW09] due to the system modifications that we made to optimize for efficiency and the faster group operations over the Bahretto-Naerig curves that were used in the MCL library.

More recently, an identity-based broadcast encryption system with efficient revocation was implemented by Ge and Wei [GW19]. The authors of the paper implemented the RIBBE system using the Charm crypto library in Python. In this paper, instead of implementing the revoking system, we focus on the adaptively secure IBBE construction given in [GW19, $\mathcal{S}$3.1]. We compare this basic IBBE scheme to other IBBE schemes from [GW09] in Section 3.7.

Among the IBBE systems, there are some interesting schemes this work did not include (due to having a comparable scheme in the study). One example by Delerablée [Del07], an adaptively secure pairings-based IBBE system with constant size ciphertexts and private keys. A more recent construction described in Ramanna-Sarkar [RS16, $\mathcal{S}$3.3] gives an adaptively secure IBBE system from the SXDH assumption.

There are various categories of broadcast encryption systems which are outside of the scope of this study. First are lattice-based broadcast encryption systems. A lattice-based IBBE system was first described in Wang-Bi [WB10]. An anonymous broadcast encryption scheme in the lattice setting was constructed by Wang, Wang and Wang [WWW15]. Recently, lattices were used to construct a broadcast and trace scheme of ciphertext size N^{ε} for any $\varepsilon > 0$ in Goyal, Quach, Waters and Wichs [GQWW19]. This is similar to Zhandry's new pairings-based broadcast and trace scheme [Zha20], which achieves a ciphertext of size N^{1-a} for any $a \in [0, 1]$ (implemented in Section 3.8.10).

Multilinear maps can also be used to construct broadcast encryption systems with optimal ciphertext sizes and short private keys, although the community currently lacks an efficient implementation of multilinear maps. Boneh and Silverberg [BS02] first use n-linear multilinear maps to construct a broadcast system secure against non-adaptive adversaries. In Boneh-Waters [BWZ14], three low-overhead broadcast encryption systems are constructed using multilinear maps.

Other categories of broadcast encryption systems require organizing users into a tree-like structure. These include revocation systems [NNL02, DF02, GST04, LSW10] and hiearchical identity-based broadcast encryption (HIBBE) systems [LLW+16]. Just like broadcast systems that rely on lattices and multilinear maps, these systems have been widely studied, but not widely implemented. Future works might expand on our study by implementing broadcast encryption systems that fall into one of these categories.

2 Preliminaries

We define the algebraic settings and security notions referenced in this work.

2.1 Pairings

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be groups of prime order p. A map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an admissible *pairing* (also called a *bilinear map*) if it satisfies the following three properties:

1. Bilinearity: for all $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_p$, it holds that $e(g^a, h^b) = e(g^b, h^a) = e(g, h)^{ab}$.
2. Non-degeneracy: if g_1 and g_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, resp., then $e(g_1, g_2)$ is a generator of $\mathbb{G}_T$.
3. Efficiency: there exists an efficient method that given any $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, computes $e(g_1, g_2)$.

A pairing generator PGen is an algorithm that on input a security parameter 1^λ, outputs the parameters for a pairing group $(p, g_1, g_2, g_T, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ such that $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are groups of prime order $p \in \Theta(2^\lambda)$ where g_1 generates $\mathbb{G}_1$, g_2 generates $\mathbb{G}_2$ and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an admissible pairing. The above pairing is called an *asymmetric* or Type-III pairing. In Type-II pairings, there exists an efficient isomorphism ψ from $\mathbb{G}_1$ to $\mathbb{G}_2$ or such an isomorphism ϕ from $\mathbb{G}_2$ to $\mathbb{G}_1$ but not both. In *symmetric* or Type-I pairings, efficient isomorphisms ψ and ϕ both exist, and thus we can consider it as though $\mathbb{G}_1 = \mathbb{G}_2$.

2.2 Broadcast Encryption

In this Section, we recall the definition of Broadcast Encryption, which is formally introduced by Fiat and Naor [FN93]. The $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ algorithms are defined as follows.

- $\mathsf{Setup}(1^\lambda, 1^N) \to (\mathsf{pk}, \mathsf{msk})$. The setup algorithm takes as input security parameter λ and number of users N. It outputs a public key pk and a master secret key msk.
- $\mathsf{KeyGen}(\mathsf{msk}, i) \to \mathsf{sk}_i$. The keygen algorithm takes as input a master secret key msk and a user index $i \in [N]$ and outputs a secret key sk_i for user i.
- $\mathsf{Enc}(\mathsf{pk}, S, m) \to \mathsf{ct}$. The encryption algorithm takes as input public key pk, a set of users $S \subseteq [N]$, a message m, and outputs a ciphertext ct.
- $\mathsf{Dec}(i, \mathsf{sk}_i, \mathsf{pk}, S, \mathsf{ct}) \to m/\perp$. The decryption algorithm takes as input an index i, secret key for i^{th} user sk_i, public key pk, a set of users $S \subseteq [N]$, a cipheretxt ct and outputs a message m or $\perp$.

Correctness. A Broadcast Encryption scheme is said to be correct if for every security parameter $\lambda \in \mathbb{N}$, any number of users $N \in \mathbb{N}$, any message $m \in \{\mathcal{M}\}_\lambda$, any subset of users $S \subseteq [N]$, any user $i \in S$, any key tuple $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$, $\mathsf{sk}_i \leftarrow \mathsf{KeyGen}(\mathsf{msk}, i)$ and $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, S, m)$, we have

$$\mathsf{Dec}(i, \mathsf{sk}_i, \mathsf{pk}, S, \mathsf{ct}) = m.$$

Definition 2.1 (Static Security)**.** *We say that a broadcast scheme satisfies static security if for every stateful Probabilistic Polynomial Time (PPT) adversary $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds*

$$\Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{pk}, \mathsf{ct}) = b \ : \ \begin{array}{c}(1^N, S^* \subseteq [N]) \leftarrow \mathcal{A}(1^\lambda);\\ (\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(\lambda, 1^N);\\ (m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{pk});\\ b \leftarrow \{0,1\}; \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, S^*, m_b)\end{array}\right] \leq \frac{1}{2} + negl(\lambda).$$

The adversary is required not to issue any secret key oracle query on any user in the set S^*.

Definition 2.2 (Semi-Static Security)**.** *We say that a broadcast scheme satisfies semi-static security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds*

$$\Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{pk}, \mathsf{ct}) = b \ : \ \begin{array}{c}(1^N, S^* \subseteq [N]) \leftarrow \mathcal{A}(1^\lambda);\\ (\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(\lambda, 1^N);\\ (\tilde{S} \subseteq S^*, m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{pk});\\ b \leftarrow \{0,1\}; \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, \tilde{S}, m_b)\end{array}\right] \leq \frac{1}{2} + negl(\lambda).$$

The adversary is required not to issue any secret key oracle query on any user in the set S^*.

Definition 2.3 (Adaptive Security)**.** *We say that a broadcast scheme satisfies adaptive security if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds*

$$\Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{pk},\mathsf{ct}) = b \;:\; \begin{array}{c} 1^N \leftarrow \mathcal{A}(1^\lambda); \\ (\mathsf{pk},\mathsf{msk}) \leftarrow \mathsf{Setup}(\lambda, 1^N); \\ (m_0, m_1, S^*) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{pk}); \\ b \leftarrow \{0,1\}; \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, S^*, m_b) \end{array}\right] \leq \frac{1}{2} + negl(\lambda).$$

The adversary is required not to issue any secret key oracle query on any user in the set S^*.

Identity-Based Broadcast Encryption. This notion is first defined in [Del07] (and independently by [SF07]) and is analogous to Broadcast Encryption, except that each user is identified by an identity string instead of an integer from the set $[N] = \{1, 2, \cdots, N\}$. In this primitive, the setup algorithm samples a master public key mpk and a master secret key msk. There is a KeyGen algorithm which takes an identity string id as input and outputs a secret key $\mathsf{sk}_{\mathsf{id}}$ for the identity. The encryption and decryption syntax remains similar to Broadcast Encryption except that they take an identity string id as input instead of an integer from the set $[N]$. The security definitions also analogous to Broadcast Encryption except that the adversary outputs a set S^* of identity strings and the oracle $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ takes identity strings as input.

2.3 Broadcast and Trace

In this section, we formally define Broadcast and Trace system and describe its security properties. The security definition is motivated by a recent work by Goyal et al. [GKW17] which points out problems with previously proposed notions of traitor tracing and proposes an indistinguishability based security definiton for the primitive.

- $\mathsf{Setup}(1^\lambda, 1^N) \rightarrow (\mathsf{pk}, \mathsf{msk})$. The setup algorithm takes as input a security parameter λ and number of users N. It outputs a public key pk, and a master secret key msk.
- $\mathsf{KeyGen}(\mathsf{msk}, i) \rightarrow \mathsf{sk}_i$. The keygen algorithm takes as input a master secret key msk and a user index $i \in [N]$ and outputs a secret key sk_i for user i.
- $\mathsf{Enc}(\mathsf{pk}, S, m) \rightarrow \mathsf{ct}$. The encryption algorithm takes as input public key pk, a set $S \subseteq [N]$ of users, a message m and outputs a ciphertext ct.
- $\mathsf{Dec}(i, \mathsf{sk}_i, \mathsf{pk}, S, \mathsf{ct}) \rightarrow m/\perp$. The decryption algorithm takes as input an index $i \in [N]$, secret key of i^{th} user, public key pk, a set of users $S \subseteq [N]$, a ciphertext ct and outputs either a message m or $\perp$.
- $\mathsf{Trace}^D(\mathsf{pk}, S_D, m_0, m_1, 1^{1/\epsilon}) \rightarrow S^*$. The tracing algorithm takes as input a public key pk, a set of users S_D, two messages m_0, m_1 and parameter $\epsilon < 1$. The algorithm has a black-box access to the decoder D and outputs a set of indices $S^* \subseteq [N]$.

Correctness. The Broadcast and Trace system is said to be correct if for every $\lambda \in \mathbb{N}$, any number of users $N \in \mathbb{N}$, every subset of users $S \subseteq [N]$, every message $m \in \mathcal{M}_\lambda$, every user $i \in S$, $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$, $\mathsf{sk}_i \leftarrow \mathsf{KeyGen}(\mathsf{msk}, i)$ and $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, S, m)$, we have

$$\mathsf{Dec}(i, \mathsf{sk}_i, \mathsf{pk}, S, \mathsf{ct}) = m.$$

Security. Intuitively, the system is said to be secure if it is IND-CPA secure and if no poly-time adversary can produce a decoder that can fool the tracing algorithm. We formally define both of these properties below.

Definition 2.4 (IND-CPA security)**.** *We say that a Broadcast and Trace scheme is* $\mathsf{IND-CPA}$ *secure if for every stateful PPT adversary* $\mathcal{A}$*, there exists a negligible function* $negl(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*, the following holds*

$$\Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{ct}) = b \ : \ \begin{array}{c} 1^N \leftarrow \mathcal{A}(1^\lambda); (\mathsf{pk},\mathsf{msk}) \leftarrow \mathsf{Setup}(\lambda, 1^N); \\ (S', m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{pk}); b \leftarrow \{0,1\}; \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, S', m_b) \end{array}\right] \leq \frac{1}{2} + negl(\lambda).$$

Let the set of indices queried by the adversary to the oracle be $S \subseteq [N]$*. Then the adversary is restricted to output the challenge set* S' *such that* $S' \subseteq [N] \setminus S$*.*

Definition 2.5 (IND-secure Traitor Tracing)**.** *Let* (Setup,Enc,Dec,Trace) *be a Broadcast and Trace scheme. For any non-negligible function* $\epsilon(\cdot)$ *and stateful PPT adversary* $\mathcal{A}$*, consider the experiment* $\mathsf{Expt\text{-}BT}_{\mathcal{A},\epsilon}(\lambda)$ *defined as follows.*

Experiment $\mathsf{Expt\text{-}BT}_{\mathcal{A},\epsilon}(\lambda)$

- $1^N \leftarrow \mathcal{A}(1^\lambda)$.
- $(\mathsf{pk},\mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$.
- $(D, S_D, m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{pk})$.
- $S^* \leftarrow \mathsf{Trace}^D(\mathsf{pk}, S_D, m_0, m_1, 1^{1/\epsilon(\lambda)})$.

Let S be the set of indices queried by $\mathcal{A}$.

Figure 1: Experiment Expt-BT

In order to define the security of tracing mechanism, we define the following events and probabilities as a function of security parameter λ*.*

- $\mathsf{Good\text{-}Dec}_{\mathcal{A},\epsilon} : \Pr[D(\mathsf{ct}) = b : b \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, S_D, m_b)] \geq 1/2 + \epsilon(\lambda)$
  $\mathsf{Pr\text{-}Good\text{-}Dec}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Good\text{-}Dec}_{\mathcal{A},\epsilon}]$
- $\mathsf{Correct\text{-}Tr}_{\mathcal{A},\epsilon} : |S^*| > 0, S^* \subseteq S \cap S_D$
  $\mathsf{Pr\text{-}Correct\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{Correct\text{-}Tr}_{\mathcal{A},\epsilon}]$
- $\mathsf{False\text{-}Tr}_{\mathcal{A},\epsilon} : S^* \not\subseteq S \cap S_D$
  $\mathsf{Pr\text{-}False\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) = \Pr[\mathsf{False\text{-}Tr}_{\mathcal{A},\epsilon}]$

The Broadcast and Trace scheme is said to have Ind-secure tracing mechanism if for every stateful PPT adversary $\mathcal{A}$*, polynomial* $q(\cdot)$ *and non-negligible function* $\epsilon(\cdot)$*, there exists negligible functions* $negl_1(\cdot)$ *and* $negl_2(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *satisfying* $\epsilon(\lambda) > 1/q(\lambda)$*,* $\mathsf{Pr\text{-}Correct\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \geq \mathsf{Pr\text{-}Good\text{-}Dec}_{\mathcal{A},\epsilon}(\lambda) - negl_1(\lambda)$ *and* $\mathsf{Pr\text{-}False\text{-}Tr}_{\mathcal{A},\epsilon}(\lambda) \leq negl_2(\lambda)$*.*

2.4 Augmented Broadcast Encryption

In this section, we recall the definition of Augmented Broadcast Encryption (AugBE) and its security properties. The notion was formally introduced by Boneh et. al. in [BSW07] as an intermediate primitive to construct Broadcast and Trace system.

- $\mathsf{Setup}(1^\lambda, 1^N) \rightarrow (\mathsf{pk},\mathsf{msk})$. The setup algorithm takes as input security parameter λ and number of users N. It outputs a public key pk and a master secret key msk.
- $\mathsf{KeyGen}(\mathsf{msk}, i) \rightarrow \mathsf{sk}_i$. The keygen algorithm takes as input a master secret key msk and a user index $i \in [N]$ and outputs a secret key sk_i for user i.
- $\mathsf{Enc}(\mathsf{pk}, S, m, \mathsf{ind}) \rightarrow \mathsf{ct}$. The encryption algorithm takes as input public key pk, a set of users $S \subseteq [N]$, a message m, and an index $\mathsf{ind} \in [N+1]$. It outputs a ciphertext ct.
- $\mathsf{Dec}(i, \mathsf{sk}_i, \mathsf{pk}, S, \mathsf{ct}) \rightarrow m/\perp$. The decryption algorithm takes as input an index i, secret key for i^{th} user sk_i, public key pk, a set of users $S \subseteq [N]$, a cipheretxt ct and outputs a message m or $\perp$.

Correctness. An AugBE scheme is said to be correct if for every security parameter $\lambda \in \mathbb{N}$, any number of users $N \in \mathbb{N}$, any message $m \in \mathcal{M}_\lambda$, any subset of users $S \subseteq [N]$, any index $\mathsf{ind} \in [N]$, any $i \in S \cap \{\mathsf{ind}, \mathsf{ind}+1, \ldots, N\}$, $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$, $\mathsf{sk}_i \leftarrow \mathsf{KeyGen}(\mathsf{msk}, i)$, and $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, S, m, \mathsf{ind})$, we have

$$\mathsf{Dec}(i, \mathsf{sk}_i, \mathsf{pk}, S, \mathsf{ct}) = m.$$

Security. We need AugBE to satisfy 2 security properties. The first is *message hiding* property which states that no PPT adversary can distinguish between encryptions of m_0 and m_1 encrypted using the last index $N+1$. The second is *index hiding* property which states that ciphertexts encrypted to index ind do not reveal any non-trivial information about the index. We formally define the security properties below.

Definition 2.6 (Message Hiding)**.** *We say that an AugBE scheme satisfies message hiding property if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds*

$$\Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{ct}) = b \;:\; \begin{array}{c} 1^N \leftarrow \mathcal{A}(1^\lambda); (\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(\lambda, 1^N); \\ (S', m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{pk}); \\ b \leftarrow \{0,1\}; \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, S', m_b, N+1) \end{array}\right] \leq \frac{1}{2} + negl(\lambda).$$

Definition 2.7 (Index Hiding)**.** *We say that an AugBE scheme satisfies index hiding property if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds,*

$$\Pr\left[\mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{ct}) = b \;:\; \begin{array}{c} (1^N, \mathsf{ind}) \leftarrow \mathcal{A}(1^\lambda); (\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(\lambda, 1^N); \\ (S', m) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{pk}); \\ b \leftarrow \{0,1\}; \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, S', m, \mathsf{ind}+b) \end{array}\right] \leq \frac{1}{2} + negl(\lambda).$$

Let the set of keys queried by the adversary be S. We restrict the adversary to satisfy $\mathsf{ind} \notin S' \vee \mathsf{ind} \notin S$.

2.5 Traitor Tracing

In this Section, we recall the definition of Traitor Tracing and its security properties. The notion was formally introduced by [CFN94]. More formal definitions were provided later by [BSW06, GKRW18]. Traitor Tracing can be considered as a special case of Broadcast and Trace, where one can encrypt a message only to the set $\{1, 2, \cdots, N\}$ consisting of all the users. Formally,

- $\mathsf{Setup}(1^\lambda, 1^N) \to (\mathsf{pk}, \mathsf{msk})$. The setup algorithm takes as input a security parameter λ and number of users N. It outputs a public key pk, and a master secret key msk.
- $\mathsf{KeyGen}(\mathsf{msk}, i) \to \mathsf{sk}_i$. The keygen algorithm takes as input a master secret key msk and a user index $i \in [N]$ and outputs a secret key sk_i for user i.
- $\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$. The encryption algorithm takes as input public key pk, a message m and outputs a ciphertext ct.
- $\mathsf{Dec}(i, \mathsf{sk}_i, \mathsf{pk}, \mathsf{ct}) \to m/\perp$. The decryption algorithm takes as input an index $i \in [N]$, secret key of i^{th} user, public key pk, a ciphertext ct and outputs either a message m or $\perp$.
- $\mathsf{Trace}^D(\mathsf{pk}, m_0, m_1, 1^{1/\epsilon}) \to S^*$. The tracing algorithm takes as input a public key pk, two messages m_0, m_1 and parameter $\epsilon < 1$. The algorithm has a black-box access to the decoder D and outputs a set of indices $S^* \subseteq [N]$.

Correctness. The Traitor Tracing system is said to be correct if for every $\lambda \in \mathbb{N}$, any number of users $N \in \mathbb{N}$, every subset of users $S \subseteq [N]$, every message $m \in \mathcal{M}_\lambda$, every user $i \in \{1, 2, \cdots, N\}$, $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$, $\mathsf{sk}_i \leftarrow \mathsf{KeyGen}(\mathsf{msk}, i)$ and $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m)$, we have

$$\mathsf{Dec}(i, \mathsf{sk}_i, \mathsf{pk}, \mathsf{ct}) = m.$$

Security. We need a Traitor Tracing scheme to satisfy 2 security properties – IND-CPA security and IND-Secure Traitor Tracing. The definitions are very similar to that of Broadcast and Trace primitive. IND-CPA security property says that a PPT adversary cannot distinguish between encryptions of 2 messages m_0, m_1 when it doesn't have access to any secret key sk_i. IND-Secure Traitor Tracing says that a PPT cannot produce a decoder D having advantage $\epsilon(\lambda)$ in distinguishing encryptions of m_0, m_1 without being detected by the Trace Algorithm. The formal definition follows similar to Definition 2.5 except that the adversary doesn't output the set S_D.

2.6 Private Linear Broadcast Encryption

In this Section, we recall the definition of Private Linear Broadcast Encryption and its security properties. The notion was formally introduced by [BSW06] as an intermediate primitive that is helpful to constructing a Traitor Tracing systems. Private Linear Broadcast Encryption can be considered as a special case of Augmented Broadcast Encryption, where one can encrypt a message only to the set $\{1, 2, \cdots, N\}$ consisting of all the users. Formally,

- $\mathsf{Setup}(1^\lambda, 1^N) \to (\mathsf{pk}, \mathsf{msk})$. The setup algorithm takes as input security parameter λ and number of users N. It outputs a public key pk and a master secret key msk.
- $\mathsf{KeyGen}(\mathsf{msk}, i) \to \mathsf{sk}_i$. The keygen algorithm takes as input a master secret key msk and a user index $i \in [N]$ and outputs a secret key sk_i for user i.
- $\mathsf{Enc}(\mathsf{pk}, m, \mathsf{ind}) \to \mathsf{ct}$. The encryption algorithm takes as input public key pk, a set of users $S \subseteq [N]$, a message m, and outputs a ciphertext ct.
- $\mathsf{Dec}(i, \mathsf{sk}_i, \mathsf{pk}, \mathsf{ct}) \to m/\perp$. The decryption algorithm takes as input an index i, secret key for i^{th} user sk_i, public key pk, a cipheretxt ct and outputs a message m or $\perp$.

Correctness. A PLBE scheme is said to be correct if for every security parameter $\lambda \in \mathbb{N}$, any number of users $N \in \mathbb{N}$, any message $m \in \mathcal{M}_\lambda$, any index $\mathsf{ind} \in [N]$, any $i \in \{\mathsf{ind}, \mathsf{ind}+1, \ldots, N\}$, $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$, $\mathsf{sk}_i \leftarrow \mathsf{KeyGen}(\mathsf{msk}, i)$, and $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, S, m, \mathsf{ind})$, we have

$$\mathsf{Dec}(i, \mathsf{sk}_i, \mathsf{pk}, \mathsf{ct}) = m.$$

Security. We need a PLBE scheme to satisfy 2 security properties – Message Hiding and Index Hiding properties. Message Hiding property says that a PPT adversary cannot distinguish between encryptions of 2 messages m_0, m_1 when encrypted using the last index $N+1$. Index Hiding property says that a PPT adversary cannot distinguish between encryptions of a message m encrypted using any 2 indices i and $i+1$. The formal definitions are very similar to that of Augmented Broadcast Encryption primitive, except that the adversary doesn't output the set S' and encryption algorithm doesn't take the set S' as input.

3 Broadcast Encryption: Implementations and Comparisons

We provide a thorough implementation and comparison of various broadcast encryption systems. We compare public-key broadcast encryption systems, identity-based broadcast encryption (IBBE) systems, and trace-and-revoke systems. All of these schemes are implemented in the asymmetric pairing setting using the MCL pairings library. All of the pairings-based schemes are tested over the Barreto-Naehrig curve BN254. This curve has 110 bit security. Group elements in $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ occupy 32 bytes, 64 bytes, and 381 bytes of space in memory, respectively. Elements in $\mathbb{Z}_p$ occupy 254 bits of space. For schemes that were originally written in the symmetric pairing setting, we translate them into the asymmetric pairing setting and describe our constructions. We also compare all of the systems to an ElGamal baseline system, which is described in detail in 3.2. We implement the baseline system over prime-order elliptic curve groups in OpenSSL since it

does not require pairings. Note that the baseline system was implemented using C++ but the other schemes were implemented using Java.

We compare the setup, encryption, key generation, and decryption times in each of our systems. The runtimes are tested by setting the size of the subset of privileged users S to be equal to some percent of the total number of users in the system. This ensures that the subset size scales with the number of users in the system. All of the runtimes were tested on a 2014 Macbook Air with a 1.4 GHz Dual-Core Intel Core i5 processor and 4GB RAM.

We also compare the sizes of the public key, private key, and ciphertext for each of the systems. Table 1 shows how the sizes scale using big-O notation. It also provides an overview of the systems that we implement.

The source code for our implementation can be viewed here. At the top of each file we include comments that describe slight changes we made to each system, which we also discuss throughout this section.

Scheme	Type	Ciphertext Size	Private Key Size	Public Key Size	Security
ElGamal baseline	Public Key Encryption	$O(\lambda \cdot \lvert S \rvert)$	$O(\lambda)$	$O(\lambda \cdot N)$	CPA-secure
[BGW05, $\mathcal{S}$3.1]	Broadcast Encryption	$O(\lambda)$	$O(\lambda)$	$O(\lambda \cdot N)$	static attacker
[BGW05, $\mathcal{S}$3.2]	Broadcast Encryption	$O(\lambda \cdot \sqrt{N})$	$O(\lambda)$	$O(\lambda \cdot \sqrt{N})$	static attacker
[Wat09]	Broadcast Encryption	$O(\lambda)$	$O(\lambda \cdot N)$	$O(\lambda \cdot N)$	adaptive attacker
[GW09, $\mathcal{S}$3.1]	Broadcast Encryption	$O(\lambda)$	$O(\lambda \cdot N)$	$O(\lambda \cdot N)$	semi-static attacker
[GW09, $\mathcal{S}$4.1]	IBBE	$O(\lambda \cdot \ell)$	$O(\lambda)$	$O(\lambda \cdot \ell)$	adaptive attacker
[GW09, $\mathcal{S}$4.3.1]	IBBE	$O(\lambda)$	$O(\lambda)$	$O(\lambda \cdot \ell)$	semi-static attacker
[GW19, $\mathcal{S}$3.1]	IBBE	$O(\lambda)$	$O(\lambda \cdot N)$	$O(\lambda \cdot N)$	adaptive attacker
[Zha20, $\mathcal{S}$9.3]	Risky Broadcast and Trace	$O(\lambda \cdot N)$	$O(\lambda)$	$O(\lambda)$	adaptive attacker
[GKSW10, $\mathcal{S}$5.2]	Broadcast and Trace	$O(\lambda \cdot \sqrt{N})$	$O(\lambda \cdot \sqrt{N})$	$O(\lambda \cdot \sqrt{N})$	adaptive attacker and public tracing
[Zha20, $\mathcal{S}$9.3]	Broadcast and Trace	$O(\lambda \cdot N^{1-a})$	$O(\lambda \cdot N^{1-a})$	$O(\lambda \cdot N^{a})$	adaptive attacker
[GKSW09]	PLBE (traitor-tracing)	$O(\lambda \cdot \sqrt{N})$	$O(\lambda)$	$O(\lambda \cdot \sqrt{N})$	private tracing

Table 1: A summary of the pairing-based broadcast encryption systems that we implemented. Let N be the number of users in the broadcast system, ℓ be the maximal size of the subset of users S such that $|S| \leq \ell$, and λ be the security parameter. Note that for the broadcast and trace system [Zha20, $\mathcal{S}$9.3], $a \in [0, 1]$.

3.1 Fundamental Operations

Every broadcast system that was implemented in this paper relies on a set of fundamental operations. By comparing the efficiencies of these basic operations, we can later use this information to explain the runtimes of the broadcast encryption systems which we obtained from our implementation. In Table 2, we compare the efficiencies of basic operations over different groups that we use in our implementation. In this table, $\mathbb{G}$ is the elliptic curve group NIST P-256 used to implement the ElGamal baseline scheme in 3.2. $\mathbb{G}_1$, $\mathbb{G}_2$, and

$\mathbb{G}_T$ are pairing groups over the curve BN254, which we use to implement all of the pairings-based schemes. We say that the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ have prime order p, and $\mathbb{Z}_p$ is the corresponding finite field.

For $\mathbb{G}_1$ and $\mathbb{G}_2$, "random sampling" is the time to choose a random generator in the specified group using an efficient function which hashes and maps an array of bytes to an element in the target group. For $\mathbb{G}_T$, random sampling is defined as the amount of time to apply the pairing operation to calculate $e(g_1, g_2) = g_T \in \mathbb{G}_T$. For $\mathbb{Z}_p$, random sampling is the amount of time required to choose a random $\alpha \in \mathbb{Z}_p$ using a cryptographically secure pseudo random number generator (CSPRNG). For $\mathbb{G}$, $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ a "multiplication" is defined as the time for a single group operation over the target group. For $\mathbb{Z}_p$, a "multiplication" is the time to actually multiply together two values over the finite field. Finally, let g be a generator in the target group $\mathbb{G}$, $\mathbb{G}_1$, $\mathbb{G}_2$, or $\mathbb{G}_T$. Then an exponentiation g^α is the time for α applications of the group operation over this group. We do not measure the time for exponentiating over $\mathbb{Z}_p$ because exponentiation times vary depending on the value of the exponent. Additionally, exponentiation over $\mathbb{Z}_p$ is not relevant in any of the schemes that we implement.

To avoid potential precision errors, we run each operation ten thousand times instead of measuring operation runtimes individually. Key takeaways from Table 2 are as follows:

1. All operations are faster over $\mathbb{G}_1$ than over $\mathbb{G}_2$.
2. Group multiplications are much faster than group exponentiations.
3. Random sampling takes a small amount of time over $\mathbb{G}$ and $\mathbb{Z}_p$, while it is much more costly over the pairing curves $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$.
4. The most costly individual operation is the pairing operation $e(g_1, g_2)$.

By comparing the runtimes of the underlying operations, we can tailor each of our schemes to achieve maximum possible efficiency. Whether this means simply switching the elements in $\mathbb{G}_1$ and $\mathbb{G}_2$ or precomputing a pairing in the setup phase, we can always modify a scheme to achieve the best possible runtimes.

Item	Random Sampling	Multiplication	Exponentiation
$\mathbb{G}$	5.99 ms	26.69 ms	191.59 ms
$\mathbb{G}_1$	227.31 ms	10.14 ms	950.79 ms
$\mathbb{G}_2$	1.27 s	32.26 ms	1.93 s
$\mathbb{G}_T$	6.48 s	27.66 ms	3.70 s
$\mathbb{Z}_p$	6.48 ms	1.46 ms	—

Table 2: Time costs for ten thousand evaluations of each operation. Let ms denote milliseconds and s denote seconds.

3.2 The ElGamal Baseline Scheme

We begin by constructing a simple baseline system: ElGamal encryption [Gam85] with shared parameters. Our baseline system is CPA-secure and does not require pairings. Private key sizes are constant, but the public key size grows linearly with the number of users in the system.

Setup(λ, n)***:*** Let $\mathbb{G}$ be an elliptic curve group of prime order p for which the DDH assumption holds true. Pick a random generator $g \in \mathbb{G}$. For $i = 1, 2, \ldots, n$ pick a random $x_i \in \mathbb{Z}_p$ and compute $h_i = g^{x_i}$. The

master public key is $PK = (g, h_1, ..., h_n)$ and the private key for user i is x_i. Output the public key PK and the n private keys $x_1, x_2, \ldots, x_n$.

Enc$(PK,\ S,\ m)$***:*** To encrypt a message m to a set of users S, first pick a random $y \in \mathbb{Z}_p$. For each user $i \in S$, compute $z_i = (h_i)^y \cdot m$. Output the ciphertext $CT = (g^y, z_1, \ldots, z_S)$.

Dec$(i,\ CT)$***:*** Parse the ciphertext as $CT = (c, z_1, \ldots, z_S)$. User i decrypts by computing $m = z_i \ / \ (c^{x_i})$.

We implement this baseline scheme and test it in OpenSSL over the curves NIST P-192, NIST P-224, NIST P-256, NIST P-384, and NIST P-521. Based on the results from our implementation, we use the results over the curve NIST P-256 as a basis of comparison to the pairing-based scheme runtimes over BN254. The runtimes are the fastest over this curve and the 128 bit security provided by NIST P-256 is very close to the 110 bit security provided by BN254. In Table 3, we include runtimes for the baseline scheme when the number of users in the system $N = 100$K and the size of the subset of users $|S| = 10$K.

Curve	Security	Setup Time	Encrypt Time	Decrypt Time
NIST P-192	96 bits	39.27 s	3.78 s	0.51 ms
NIST P-224	112 bits	5.51 s	515.13 ms	0.09 ms
NIST P-256	128 bits	2.40 s	248.08 ms	0.05 ms
NIST P-384	192 bits	145.82 s	14.66 s	2.67 ms
NIST P-521	260.5 bits	37.04 s	3.62 s	0.73 ms

Table 3: Runtimes for the ElGamal baseline over different curves. Let s denote seconds and ms denote milliseconds.

3.3 Boneh-Gentry-Waters Scheme Using Asymmetric Pairings

The Boneh-Gentry-Waters-Scheme, [BGW05], refers to a fully collusion resistant public key broadcast system for stateless receivers. In the paper, two schemes are described, and both are secure against static adversaries. In the "special case", [BGW05, $\mathcal{S}3.1$], the public key grows linearly with the total number of users in the broadcast system. Ciphertext sizes and private key sizes are constant. In the general construction [BGW05, $\mathcal{S}3.2$], the public key and ciphertext are both of size $O(\lambda \cdot \sqrt{N})$, and private key sizes are constant. We rewrite both of these schemes using Type-III pairings, strategically placing certain group elements in $\mathbb{G}_1$ and $\mathbb{G}_2$ to optimize the efficiency of our construction. We also add a KeyGen function that generates the private key for an individual user in the system instead of generating the private keys for all N users in the Setup phase. This facilitates the comparison of this scheme to other public-key broadcast encryption schemes which include KeyGen functions, as we will later see in Section 3.6.

3.3.1 A Special Case

Setup$(\lambda,\ n)$***:*** The setup algorithm takes as input the security parameter λ and the number of users n in the system. Generate two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p with a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Choose random generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. Pick a random $\alpha \in \mathbb{Z}_p$ and compute $h_i = {g_1}^{\alpha^i} \in \mathbb{G}_1$ for $i = 1, 2, \ldots, n, n+2, \ldots, 2n$. Compute $u_i = {g_2}^{\alpha^i} \in \mathbb{G}_2$ for $i = 1, 2, \ldots, n$. Choose a random $\gamma \in \mathbb{Z}_p$ and set $v = {g_1}^{\gamma}$. The public key is:

$$PK = (g_1, g_2, h_1, h_2, \ldots, h_n, h_{n+2}, \ldots, h_{2n}, u_1, u_2, \ldots, u_n, v) \in {\mathbb{G}_1}^{2n+1} \times {\mathbb{G}_2}^{n+1}$$

The master secret key is defined as $MSK = \{\gamma, PK\}$. Output the public key PK and the master secret key MSK.

KeyGen(i, MSK)***:*** The secret key for user i is $d_i = {h_i}^{\gamma} \in \mathbb{G}_1$. Output d_i.

Enc(S, PK)***:*** To encrypt to a subset of users S, choose a random $t \in \mathbb{Z}_p$ and set $K = e(h_{n+1}, g_2)^t \in \mathbb{G}_T$. Note that the pairing $e(h_{n+1}, g_2)$ can be precomputed as $e(h_n, u_1)$ during the setup phase. Next, set

$$\text{Hdr} = \left({g_2}^t, (v \cdot \prod_{j \in S} h_{n+1-j})^t \right)$$

and output the pair (Hdr, K).

Dec$(i, d_i, PK, S, \text{Hdr})$***:*** Let Hdr $= (C_0, C_1)$ and recall that $d_i \in \mathbb{G}_1$. Then, output

$$K = e(C_1, u_i) \ / \ e(d_i \cdot \prod_{\substack{j \in S \\ j \neq i}} h_{n+1-j+i}, C_0)$$

In our implementation, we precompute $K = e(h_{n+1}, g_2)$ during the setup phase so that encryption does not require pairings. We save this value in our system and then during encryption, raise it to the power of a random $t \in \mathbb{Z}_p$. Also notice that in our construction, we choose $h_i \in \mathbb{G}_1$ and $u_i \in \mathbb{G}_2$ to maximize the efficiency of our system. Since the $\mathbb{G}_1$ group size is smaller than the $\mathbb{G}_2$ group size, the multiplication and exponentiation group operations are faster over $\mathbb{G}_1$. In the computation of the public key, we calculate a total of $2n$ exponentiations over $\mathbb{G}_1$ but only n exponentiations over $\mathbb{G}_2$. Likewise, during encryption and decryption, we calculate $|S| - 1$ and $|S| - 2$ group multiplications, respectively. To maximize the efficiency of our system these group multiplications are all computed over $\mathbb{G}_1$.

In Table 4, we present the encryption times for our described construction for varying subset sizes. We define the subset size to be some percent of the total number of users in the system. Notice a general trend that as the subset size percentage increases, so does the encryption time. This is because even though the ciphertext sizes are constant, $O(|S|)$ multiplications over $\mathbb{G}_1$ are required to compute the product $v \cdot \prod_{j \in S}(h_{n+1-j})$ during encryption.

Subset Size	Encryption time when number of system users $N =$				
	100	1K	10K	100K	1M
1%	1.24 ms	0.88 ms	1.29 ms	6.08 ms	87.24 ms
5%	0.71 ms	1.31 ms	2.16 ms	14.06 ms	340.73 ms
10%	1.26 ms	1.32 ms	2.40 ms	16.35 ms	664.76 ms
20%	1.51 ms	1.69 ms	3.27 ms	30.89 ms	1.13 s
50%	0.75 ms	2.19 ms	8.60 ms	67.59 ms	1.62 s

Table 4: Encryption times for the Boneh-Gentry-Waters Special Case scheme. Let s denote seconds and ms denote milliseconds.

3.3.2 General Construction

We implement the general construction by setting $B = \lfloor \sqrt{n} \rfloor$ as the authors of [BGW05] suggest. In this case, B is an arbitrary parameter that scales the public key and ciphertext to the desired size, and setting B to the specified value enables us to achieve the optimal public key and ciphertext sizes of $O(\lambda \cdot \sqrt{N})$. Again, we modify this system to include a KeyGen function that generates the private key for an individual user in the system instead of generating the private keys for all N users in the Setup phase. In the asymmetric pairing setting, the B-broadcast system works as follows:

Setup$(\lambda,\, n)$***:*** The setup algorithm takes as input the security parameter λ and the number of users n in the system. Generate two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p with a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Choose random $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, and $\alpha \in \mathbb{Z}_p$. Compute $h_i = {g_1}^{\alpha^i} \in \mathbb{G}_1$ for $i = 1, 2, \ldots, B, B+2, \ldots, 2B$ and $u_i = {g_2}^{\alpha^i} \in \mathbb{G}_2$ for $i = 1, 2, \ldots, B$. Let $A = \lceil \frac{n}{B} \rceil$. Pick random $\gamma_1, \gamma_2, \ldots, \gamma_A \in \mathbb{Z}_p$ and set $v_1 = {g_1}^{\gamma_1}, v_2 = {g_1}^{\gamma_2}, \ldots, v_A = {g_1}^{\gamma_A}$. The public key is:

$$PK = (g_1, g_2, h_1, h_2, \ldots, h_B, h_{B+2}, \ldots, h_{2B}, u_1, u_2, \ldots, u_B, v_1, v_2, \ldots, v_A) \in \mathbb{G}_1^{2B+A} \times \mathbb{G}_2^{B+1}$$

Define the master secret key as $MSK = \{\gamma_1,\, \gamma_2,\, \ldots,\, \gamma_A,\, PK\}$. Output the public key PK and the master secret key MSK.

KeyGen$(i,\, MSK)$***:*** To generate the key for user i, first write i as $i = (a-1)B + b$ for some $1 \le a \le A$ and $1 \le b \le B$. More specifically, $a = \lceil \frac{i}{B} \rceil$ and $b = B$ if $i \pmod B$ equals 0, otherwise $b = i \pmod B$. The private key for user i is set as $d_i = (h_b)^{\gamma_a} \in \mathbb{G}_1$. Output d_i.

Enc$(S,\, PK)$***:*** To encrypt to a subset of users S, choose a random $t \in \mathbb{Z}_p$ and set $K = e(h_{B+1}, g_2)^t \in \mathbb{G}_T$. Then, for each $\ell = 1, 2, \ldots, A$, define the subsets $\hat{S}_\ell$ and S_ℓ as

$$\hat{S}_\ell = S \cap \{\ell B - B + 1,\, \ell B - B + 2,\, \ldots,\, \ell B\} \quad \text{and} \quad S_\ell = \{x - \ell B + B \mid x \in \hat{S}_\ell\} \subseteq \{1, 2, \ldots, B\}$$

Compute the broadcast ciphertext as

$$\text{Hdr} = \left(g_2^t,\, (v_1 \cdot \prod_{j \in S_1} h_{B+1-j})^t,\, (v_2 \cdot \prod_{j \in S_2} h_{B+1-j})^t,\, \ldots, (v_A \cdot \prod_{j \in S_A} h_{B+1-j})^t \right) \in \mathbb{G}_1^A \times \mathbb{G}_2$$

Output the pair (Hdr, K).

Dec$(i,\, d_i,\, PK,\, S,\, \text{Hdr})$***:*** Parse Hdr as $(C_0, C_1, \ldots, C_A)$ and recall that $d_i \in \mathbb{G}_1$. Write i as $i = (a-1)B + b$ for some $1 \le a \le A$ and $1 \le b \le B$. Recall the definition of S_ℓ from encryption, and calculate

$$K = e(C_a, u_b) \;/\; e(d_i \cdot \prod_{\substack{j \in S_a \\ j \neq b}} h_{B+1-j+b}, C_0)$$

In our implementation, we once again precompute the pairing part of K, $e(h_{B+1}, g_2)$, in the setup function and raise it to the power of a random $t \in \mathbb{Z}_p$ during encryption. We again optimize the scheme by choosing $h_i \in \mathbb{G}_1$ and $u_i \in G_2$. During the setup phase, we compute $2B + A - 1$ exponentiations over $\mathbb{G}_1$, but only B exponentiations over the larger group $\mathbb{G}_2$. During encryption, we compute all group multiplications $v_i \cdot \prod_{j \in S_i}(h_{B+1-j})$ for $i \in \{1, 2, \ldots, A\}$ over $\mathbb{G}_1$, and during decryption, we compute the product $d_i \cdot \prod_{\substack{j \in S_a \\ j \neq b}}(h_{B+1-j+b})$ over $\mathbb{G}_1$. This gives the most efficient possible system in the Type-III pairing setting.

In Table 5, we give the encryption times from our implementation. We notice that runtimes increase when we read the table from left to right. However, when we read the table from top to bottom, we see mixed results. In the encryption algorithm, runtimes are determined by two significant steps. First is the computation of S_ℓ, which is dominated by the number of operations required to calculate $\hat{S}_\ell = S \cap \{\ell B - B + 1, \ell B - B + 2, \ldots, \ell B\}$. In order to compute the intersection of two sets, for each item in the set $\{\ell B - B + 1, \ell B - B + 2, \ldots, \ell B\}$, the system must check if there is a corresponding item in S. Since we use hash sets to compute this intersection, the time that it takes to lookup an item in S is $O(1)$. But we still need to iterate over each item in the original set of size B, so this step will take time $O(B)$. The computation of the subset $\hat{S}_\ell$ is an intermediate step which dominates the computation of S_ℓ for $l = 1, 2, \ldots, A$. Therefore, the total time to compute all of these subsets is $O(A \cdot B) = O(N)$. After computing these subsets, there is a second step. The system still has to calculate the product $v_i \cdot \prod_{j \in S_i} (h_{B+1-j})$ for $i \in \{1, 2, \ldots, A\}$. According to our implementation, this takes a total of $|S|$ group multiplications. Hence, the overall time complexity for the encryption algorithm is given by $O(N + \lambda \cdot |S|)$. The $O(N)$ operations in computing the S_ℓ subsets are individually much less costly than each of the $|S|$ group multiplications, but they still influence runtimes to an extent. Reading the table from top to bottom, we keep the total number of users in the system N constant while increasing the subset size $|S|$. For the smaller values of $|S|$ (i.e. N = 100, 1K, 10K), the slight increase in the value of $|S|$ does not significantly affect runtimes to an extent that it can be explained by the big-O notation. But reading the table from left to right, we increase both $|S|$ *and* the value of N, which causes runtimes to increase as expected.

Subset Size	Encryption time when number of system users $N =$				
	100	1K	10K	100K	1M
1%	3.67 ms	9.59 ms	29.62 ms	77.61 ms	288.11 ms
5%	3.62 ms	8.14 ms	24.68 ms	83.09 ms	297.83 ms
10%	3.58 ms	8.61 ms	26.60 ms	86.36 ms	306.96 ms
20%	2.77 ms	6.90 ms	30.41 ms	134.18 ms	548.08 ms
50%	4.43 ms	13.09 ms	47.39 ms	145.08 ms	897.48 ms

Table 5: Encryption times for the Boneh-Gentry-Waters general scheme. Let ms denote milliseconds

3.4 Gentry-Waters: A Semi-Static Variant of the BGW System

In [GW09], Gentry and Waters introduce the notion of semi-static security, which is between static security and adaptive security. We review their definition of semi-static security in 2.2. They construct a semi-statically secure variant of [BGW05] (3.3 the Boneh-Gentry-Waters scheme). In their system, the public key and private key both grow linearly with the total number of system users, but the ciphertext sizes are constant. In this section, we reconstruct their semi-static scheme in the asymmetric (Type-III) pairing setting and briefly analyze the encryption times from our implementation. As in the previous schemes, our construction is optimized for efficiency. We give our construction as follows:

Setup(λ, n)***:*** The setup algorithm takes as input the security parameter λ and the number of users n in the system. Generate two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p with a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Pick a random $\alpha \in \mathbb{Z}_p$ and choose random generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. Also, choose random $h_1, h_2, \ldots, h_n \in \mathbb{G}_1^n$. Set the public key $PK = (g_1, g_2, e(g_1, g_2)^\alpha, h_1, h_2, \ldots, h_n)$. The master secret key is $MSK = (g_1^\alpha, g_2^\alpha)$. Output (PK, MSK).

KeyGen$(i,\ MSK)$***:*** To generate the key for user i from the master secret key MSK, choose a random $r_i \in \mathbb{Z}_p$. Then output

$$d_i = (d_{i,0},\ d_{i,1},\ \ldots,\ d_{i,n}) \quad \text{where} \quad d_{i,0} = g_2^{r_i}\ , \quad d_{i,i} = g_1^{\alpha} \cdot h_i^{r_i}, \quad \forall_{j \neq i}\ d_{i,j} = h_j^{r_i}$$

Enc$(S,\ PK)$***:*** To encrypt to a subset of users S, choose a random $t \in \mathbb{Z}_p$ and set

$$\text{Hdr} = (C_1, C_2) \quad \text{where} \quad C_1 = g_2^t\ , \quad C_2 = (\prod_{j \in S} h_j)^t$$

Set $K = e(g_1, g_2)^{\alpha \cdot t}$. Output (Hdr, K).

Dec$(i,\ d_i,\ PK,\ S,\ \text{Hdr})$***:*** If $i \in S$, parse d_i as $(d_{i,0}, d_{i,1}, \ldots, d_{i,n})$ and Hdr as (C_1, C_2). Recall that $d_{i,0} \in \mathbb{G}_2$ and for all $j \in \{1, 2, \ldots, n\}$, $d_{i,j} \in \mathbb{G}_1$. Then output

$$K = e(d_{i,i} \cdot \prod_{\substack{j \in S \\ j \neq i}} d_{i,j},\ C_1) \cdot e(C_2, d_{i,0})$$

In our implementation, we precompute a part of K, $e(g_1, g_2)^{\alpha}$, in the setup function. Then during encryption, we raise this value to the power of a random $t \in \mathbb{Z}_p$. Also note that we strategically set $h_1, h_2, \ldots, h_n \in \mathbb{G}_1^n$ to maximize the efficiency of our system. The group operations are much faster over the curve $\mathbb{G}_1$ than they are over the larger curve $\mathbb{G}_2$. During encryption, we end up computing $|S| - 1$ group multiplications and one exponentiation over $\mathbb{G}_1$ when we calculate $C_2 = (\prod_{j \in S} h_j)^t$, but only a single exponentiation over $\mathbb{G}_2$ to calculate $C_1 = (g_2)^t$. During key generation, we compute n exponentiations over $\mathbb{G}_1$ by calculating $(h_j)^{r_i}$ for $j \in \{1, 2, \ldots, n\}$. But only one exponentiation over $\mathbb{G}_2$ is required to calculate $d_{i,0} = g_2^{r_i}$. Finally, decryption requires $n-1$ group multiplications over $\mathbb{G}_1$ to compute the product $d_{i,i} \cdot \prod_{\substack{j \in S \\ j \neq i}} (d_{i,j})$.

In Table 6, we give the encryption times for the scheme. We notice a general trend that reading the table from left to right, encryption times increase. For the larger values of N, encryption times also generally increase when reading the table from top to bottom. In both of these scenarios, the subset size is increasing dramatically, which is why we see such a great increase in encryption times. Encryption time is dominated by the time required to calculate $C_2 = (\prod_{j \in S} h_j)^t$, which requires $O(|S|)$ group multiplications over $\mathbb{G}_1$.

Subset Size	Encryption time when number of system users $N =$				
	100	1K	10K	100K	1M
1%	0.64 ms	0.59 ms	1.51 ms	1.97 ms	13.00 ms
5%	0.65 ms	1.11 ms	1.53 ms	6.21 ms	62.30 ms
10%	1.18 ms	1.37 ms	1.99 ms	12.63 ms	153.56 ms
20%	0.80 ms	0.80 ms	2.95 ms	20.99 ms	200.77 ms
50%	1.04 ms	1.59 ms	5.46 ms	65.02 ms	486.97 ms

Table 6: Encryption times for the semi-static variant of the Boneh-Gentry-Waters scheme. Let ms denote milliseconds.

3.5 Waters Dual System Broadcast Encryption System

We implement a broadcast encryption system that is secure against adaptive adversaries, described in [Wat09]. We remind readers that the adaptive security provided by this scheme is stronger than the static and semi-static security of the schemes implemented in 3.3 and 3.4, respectively. In this system, the ciphertext sizes are constant, but the public key and private key sizes grow linearly with the total number of system users. This system, like the others, was originally written in the symmetric pairing setting. So we reconstruct the system in the Type-III pairing setting, strategically choosing which group elements to place in $\mathbb{G}_1$ and $\mathbb{G}_2$ to maximize the efficiency of our system:

Setup(λ, n)***:*** The setup algorithm takes as input the security parameter λ and the number of users n in the system. Generate two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p with a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Choose random generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. Additionally, choose random generators $v, v_1, v_2, w, u_1, u_2, \ldots, u_n \in \mathbb{G}_1^{n+4}$ and exponents $a_1, a_2, b, \alpha \in \mathbb{Z}_p$. Let $\tau_1 = v \cdot v_1^{a_1}$ and $\tau_2 = v \cdot v_2^{a_2}$. The public key PK and master secret key MSK are defined as

$$PK = (g_2,\ g_1^b,\ g_2^b,\ g_2^{a_1},\ g_2^{a_2},\ g_2^{b \cdot a_1},\ g_2^{b \cdot a_2},\ \tau_1,\ \tau_2,\ \tau_1^b,\ \tau_2^b,\ w,\ u_1,\ u_2,\ \ldots,\ u_n,\ e(g_1, g_2)^{\alpha \cdot a_1 \cdot b})$$

$$MSK = (g_1,\ g_1^\alpha,\ g_1^{\alpha \cdot a_1},\ v,\ v_1,\ v_2,\ PK)$$

Note that the master secret key includes the public parameters. Output (PK, MSK).

Enc(S, PK, M)***:*** To encrypt a message $M \in \mathbb{G}_T$ to a subset of users S, first choose random $s_1, s_2, t \in \mathbb{Z}_p$. Let $s = s_1 + s_2$. Then blind M as $C_0 = M \cdot (e(g_1, g_2)^{\alpha \cdot a_1 \cdot b})^{s_2}$ and create:

$$C_1 = (g_2^b)^{s_1+s_2},\ C_2 = (g_2^{b \cdot a_1})^{s_1},\ C_3 = (g_2^{a_1})^{s_1},\ C_4 = (g_2^{b \cdot a_2})^{s_2},\ C_5 = (g_2^{a_2})^{s_2},\ C_6 = \tau_1^{s_1} \cdot \tau_2^{s_2},\ C_7 = (\tau_1^b)^{s_1} \cdot (\tau_2^b)^{s_2} \cdot w^{-t},$$

$$E_1 = (\prod_{i \in S} u_i)^t,\ \ E_2 = (g_2)^t$$

Output the ciphertext $CT = (C_0, C_1, \ldots, C_7, E_1, E_2)$.

KeyGen(k, MSK)***:*** To generate the key for user k from the master secret key MSK, choose random $r_1, r_2, z_1, z_2 \in \mathbb{Z}_p$. Let $r = r_1 + r_2$. Then create:

$$D_1 = (g_1^{\alpha \cdot a_1}) \cdot v^r,\ D_2 = g_1^{-\alpha} \cdot v_1^r \cdot g_1^{z_1},\ D_3 = (g_1^b)^{-z_1},\ D_4 = v_2^r \cdot g_1^{z_2},\ D_5 = (g_1^b)^{-z_2},\ D_6 = ({g_2}^b)^{r_2},\ D_7 = (g_2)^{r_1},$$

$$K = (u_k \cdot w)^{r_1},\ \ \forall_{i \neq k}\ K_i = (u_i)^{r_1}$$

The secret key for user k is $SK_k = D_1, D_2, \ldots, D_7,\ K,\ \forall_{i \neq k}\ K_i$. Output SK_k.

Dec(SK_k, S, CT)***:*** The decryption algorithm takes as input the ciphertext CT, the set of users S that the ciphertext is encrypted to, and the secret key for the kth user. If $k \in S$, then user k will be able to recover the message M. Recall that $CT = (C_0, C_1, \ldots, C_7, E_1, E_2)$ and $SK_k = D_1, D_2, \ldots, D_7,\ K,\ \forall_{i \neq k}\ K_i$, and compute the following:

$$A_1 = \prod_{j=1}^{5} e(C_j, D_j),\ \ A_2 = e(C_6, D_6) \cdot e(C_7, D_7)$$

$$A_3 = A_1\ /\ A_2, \quad A_4 = e(E_2,\ K \cdot \prod_{\substack{i \in S \\ i \neq k}} K_i)\ /\ e(E_1, D_7)$$

Finally, recover the message by computing

$$M = C_0 \,/\, (A_3 \,/\, A_4)$$

By defining the group generators $u_1, u_2, \ldots, u_n \in \mathbb{G}_1$ we create the most efficient possible implementation of this system in the asymmetric pairing setting. In the setup phase, the random sampling of N elements in $\mathbb{G}_1$ will be much more efficient than if those elements were sampled from the group $\mathbb{G}_2$ (see Table 2). During encryption, we have $|S| - 1$ group multiplications to compute $E_1 = (\prod_{i \in S} u_i)^t$ and a constant number of exponentiations over $\mathbb{G}_2$. During key generation, $n - 1$ exponentiations are computed over $\mathbb{G}_1$ with $\forall_{i \neq k} K_i$. And finally, the $|S| - 1$ group multiplications required to calculate $K \cdot \prod_{\substack{i \in S \\ i \neq k}} K_i$ in decryption are all computed over $\mathbb{G}_1$.

In Table 7, we show the encryption times from our implementation of this scheme. We notice a consistent trend that reading the table from left to right and from top to bottom, encryption times increase. In other words, as subset size increases, so does encryption time. This agrees in theory. Encryption runtimes are dominated by the computation of $E_1 = (\prod_{i \in S} u_i)^t$, which requires $O(|S|)$ group multiplications over $\mathbb{G}_1$.

Subset Size	Encryption time when number of system users $N =$				
	100	1K	10K	100K	1M
1%	2.01 ms	2.23 ms	2.97 ms	3.35 ms	14.22 ms
5%	2.02 ms	2.57 ms	4.39 ms	7.70 ms	73.20 ms
10%	2.03 ms	2.38 ms	3.44 ms	14.55 ms	131.80 ms
20%	2.06 ms	3.26 ms	4.49 ms	22.90 ms	239.53 ms
50%	2.61 ms	3.37 ms	8.22 ms	79.07 ms	494.90 ms

Table 7: Encryption times for the Waters Dual Broadcast System. Let ms denote milliseconds.

3.6 Comparison of General Public Key Broadcast Encryption Systems

We now compare the broadcast encryption systems that we describe in 3.3, 3.4, and 3.5 to each other, and to the baseline scheme which we describe in 3.2. We perform a runtime evaluation based on experimental values for setup, encryption, key generation (when applicable), and decryption. Based on these values, we then count individual operations and construct theoretical big-O notation runtime tables for each of the functions in each scheme. We only compare the runtimes based on the runtime tables that we construct in this paper, but we refer the reader to this page to view all of the runtimes. We also do a size evaluation based on the actual sizes of the group elements in $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ over the curve BN254, given in Table 9. We have already given the theoretical sizes of the public key, private key, and ciphertext for each scheme at the beginning of this section in Table 1.

3.6.1 Setup Times

We start by analyzing the setup times presented in Table 8. For all the pairing-based schemes, the setup phase requires computing a public key PK and master secret key MSK. Computing the master secret key takes a negligible, constant amount of time, but the time that it takes to compute the public key varies. The baseline scheme setup phase requires $O(N)$ exponentiations over the elliptic curve group $\mathbb{G}$ to calculate a linear-sized PK. On the other hand, [BGW05, $\mathcal{S}$3.2] requires $O(\sqrt{N})$ exponentiations over $\mathbb{G}_1$ to calculate a public key of size $O(\lambda \cdot \sqrt{N})$. All the other pairing-based schemes require $O(N)$ exponentiations over $\mathbb{G}_1$

to calculate a linear-sized public key. Recall that group operations over the elliptic curve group $\mathbb{G}$ used for the baseline are faster than operations over $\mathbb{G}_1$ (see Table 2). This explains why the baseline is faster than all of the pairings-based schemes *except* [BGW05, $\mathcal{S}$3.2].

Notice that setup times for [BGW05, $\mathcal{S}$3.2] appear to be consistently faster than those for the baseline when $N >= 1K$. When $N = 100$, the difference in the total number of exponentiations computed during setup for the baseline and [BGW05, $\mathcal{S}$3.2] is negligible. Hence, faster setup times for the baseline can be attributed the faster time for individual exponentiations over $\mathbb{G}$ compared to $\mathbb{G}_1$. When $N >= 1K$, though, [BGW05, $\mathcal{S}$3.2] has faster setup times because calculating the public key requires much fewer exponentiations than for the baseline. Even though individual exponentiations are still faster over $\mathbb{G}$ in the baseline scheme, the sheer number of exponentiations required to calculate the public key has increased to an extent that it results in slower setup times.

3.6.2 Encryption Times

Next, we look at the encryption times in Table 8. On a first glance, it might seem surprising that the encryption times for most of the pairing-based schemes appear to be consistently faster than those for the baseline. But then if we look back at the baseline construction in 3.2, we notice that during encryption, we have to calculate $z_i = (h_i)^y \cdot m$ for each $i \in S$, in addition to g^y. Overall, this takes $|S| + 1$ group exponentiations and $|S|$ multiplications. Just like the baseline, all of the pairing-based schemes compute a part of the ciphertext with with $O(|S|)$ group multiplications. But for all of the pairing-based schemes except [BGW05, $\mathcal{S}$3.2], the total number of exponentiations computed during encryption is constant. In [BGW05, $\mathcal{S}$3.1] and [GW09, $\mathcal{S}$3.1], we only need one group exponentiation each time to compute $C_0 = (g_2)^t$. In [Wat09], we have exactly six exponentiations over $\mathbb{G}_1$ and size over $\mathbb{G}_2$ every time we compute the ciphertext. This makes the total time for encryption for these schemes less than that for the baseline as the value of N increases. We better explain the results in a series of observations:

- When $N = 100$, the baseline encryption is the most efficient, even though it requires computing more group exponentiations than the other schemes. This is because the efficient group operations over the elliptic curve group $\mathbb{G}$ used for the baseline are outweighed by the slower group operations in the pairing-based schemes. However, the number of exponentiations required for the baseline encryption increases linearly with N. So when $N >= 1K$, despite the faster group operations over $\mathbb{G}$, the number of exponentiations increases sharply for the baseline, while it stays constant for all the pairing-based schemes except [BGW05, $\mathcal{S}$3.2]. Hence, all of the pairing-based schemes except [BGW05, $\mathcal{S}$3.2] have faster encryption times when $N >= 1K$.

- If we compare the baseline to [BGW05, $\mathcal{S}$3.2], we notice that [BGW05, $\mathcal{S}$3.2] is only more efficient than the baseline when $N \geq 100K$. We recall that encryption in [BGW05, $\mathcal{S}$3.2] requires a total of $|S|$ group multiplications over $\mathbb{G}_1$, one exponentiation over $\mathbb{G}_2$, and A exponentiations over $\mathbb{G}_1$, where $A \approx \sqrt{N}$. We also recall that in encryption for [BGW05, $\mathcal{S}$3.2], we need to compute S_ℓ for $\ell \in \{1, 2, \ldots, A\}$ by computing the intersection of integer subsets. This technically takes $O(N)$ time to run, but since iterating over and adding integers to subsets is much faster than multiplying/exponentiating group elements, this step in encryption is fairly rapid. When $N = 100$, $1K$, $10K$, the baseline scheme's faster encryption times can be attributed to the efficiency of group operations over the elliptic curve group $\mathbb{G}$. Combined with the time to compute S_ℓ for $\ell \in \{1, 2, \ldots, A\}$, the $O(\sqrt{N})$ exponentiations in encryption for [BGW05, $\mathcal{S}$3.2] take longer to compute than the $O(N)$ exponentiations for the baseline. This changes when $N \geq 100K$. Now, the sheer number of exponentiations required for the baseline encryption has increased so greatly that the baseline encryption takes longer than that for [BGW05, $\mathcal{S}$3.2].

- When we compare the pairing-based schemes to each other, we see that [GW09, $\mathcal{S}$3.1] consistently has the fastest encryption times. For the smaller values of N, the only other scheme that has encryption times nearly as fast is [BGW05, $\mathcal{S}$3.1]. As N grows larger, the encryption times for [Wat09] grows

closer to those for [GW09, $\mathcal{S}$3.1] and [BGW05, $\mathcal{S}$3.1]. We recall that [GW09, $\mathcal{S}$3.1] is actually a semi-static variant of [BGW05, $\mathcal{S}$3.1]. The encryption algorithms for both of these schemes are very similar. Hence the similar runtimes. What is significant, though, is that a semi-statically secure broadcast encryption system achieved faster encryption times than its static counterpart. So far, we see that the [GW09, $\mathcal{S}$3.1] appears to be a well-performing system. It has the fastest encryption times, very fast setup times, and a moderately strong level of security. For adaptive security, [Wat09] seems to be a very good option. The only downside to both of these schemes, as we will shortly see, is their large private key sizes.

3.6.3 Key Generation Times

We now look at the times for key generation. The baseline scheme, [GW09, $\mathcal{S}$3.1], and [Wat09] are all secure against static adversaries. Private key sizes are constant, and therefore, single-user key generation times are constant. In order to achieve semi-static and adaptive security, though, the private key size must be expanded. The key generation algorithms for [GW09, $\mathcal{S}$3.1] and [Wat09] generate much larger private keys of size $O(\lambda \cdot N)$. The problem that we found in our implementation is that these key generation algorithms take very long to run. In both [GW09, $\mathcal{S}$3.1] and [Wat09], it takes more than 1.5 minutes to generate a key for a single user when the total number of system users $N = 1M$. For these two schemes, reading the key generation runtimes from left to right, we see that they increase linearly with the number of users in the system. This makes sense because it requires $O(N)$ operations over $\mathbb{G}_1$ to generate a single user's linear-sized private key. In [GW09, $\mathcal{S}$3.1], we need $N+1$ exponentiations over $\mathbb{G}_1$ to calculate the private key for a single user. Key generation for [Wat09] is similar, but slightly slower. In addition to the $N+1$ exponentiations over $\mathbb{G}_1$, the system needs to calculate $D_1, D_2, \ldots, D_7$. It would take up a lot of time and space to generate and store the private keys for a large subset of privileged users. Using big-O notation, the keys for all the privileged users in a subset S would take up $O(\lambda \cdot N \cdot |S|)$ space.

3.6.4 Decryption Times

Finally, we look at the decryption times. The only two schemes for which the decryption times remained relatively constant as the total number of system users increased were the baseline scheme and [BGW05, $\mathcal{S}$3.2]. For the baseline scheme, decryption does not require any pairings. The decryption algorithm runs in constant time because only a single division needs to be computed ($m = z_i \,/\, (c^{x_i})$), regardless of the value of N. In [BGW05, $\mathcal{S}$3.2], since we break up our broadcast encryption system into $\sqrt{N}$ instances, we only have to use a single one of those instances – which we created during encryption – to decrypt the message. It is a tradeoff: slower encryption times to calculate each of the instances, but approximately constant decryption times. All of the pairing-based schemes except [Wat09] required only two pairings to be computed during decryption. [Wat09] required nine pairings. This large number of pairings explains why the decryption times are why the decryption times are consistently the slowest for this system for $N <= 100K$. We again see that the decryption times for [BGW05, $\mathcal{S}$3.1] are very similar to those of its semi-static counterpart, [GW09, $\mathcal{S}$3.1]. And this makes sense. For both the schemes, decryption times are dominated by a step that requires $|S| - 1$ group multiplications over $\mathbb{G}_1$.

3.6.5 Overall Evaluation

If we consider key generation a step in the decryption process, then [GW09, $\mathcal{S}$3.1] and [Wat09] by far have the slowest runtimes. But we recall that these are the only two systems that are secure against non-static adversaries. It is a tradeoff: in order to achieve the higher level of security, the decryption process will be slower.

The encryption times for [GW09, $\mathcal{S}$3.1] and [Wat09] were comparable, if not better, than the encryption times for the systems which were secure against static adversaries. The setup times were faster because the n private keys were not computed during the setup phase. The only other downside with both of these systems are the long key generation times and the large private key sizes. If we can find a way to improve

the efficiency of key generation and shorten the private key size, perhaps by using a tradeoff like in [BGW05, $\mathcal{S}$3.2], then these systems would likely perform much better.

Looking at Table 9, we argue that if the primary goal of the broadcast system is to achieve short public and private key sizes and efficient decryption times, then we recommend using [BGW05, $\mathcal{S}$3.2]. This is the only scheme that achieves public key and ciphertext sizes of $O(\lambda \cdot \sqrt{N})$. The private key sizes are constant. Even though the setup times for this scheme are not more efficient than those for [GW09, $\mathcal{S}$3.1] and [Wat09], they are still very fast in comparison to [BGW05, $\mathcal{S}$3.1]. Additionally, the public key and private keys in [GW09, $\mathcal{S}$3.1] and [Wat09] are all of size $O(\lambda \cdot N)$. This is very large. But we recall that while [BGW05, $\mathcal{S}$3.1] and [BGW05, $\mathcal{S}$3.2] are secure only against static adversaries, [GW09, $\mathcal{S}$3.1] is secure against semi-static adversaries and [Wat09] is secure against adaptive adversaries. If we judge these schemes only by the efficiency of their decryption times and public/private key sizes, and we desire a stronger level of security, then we recommend [GW09]. In general, the decryption times for [GW09, $\mathcal{S}$3.1] are much faster because they only require two applications of the pairing algorithm, while [Wat09] requires nine pairings in decryption. Nevertheless, as the total number of system users N grows larger, the decryption times for [Wat09] approach the times for [GW09, $\mathcal{S}$3.1]. So if we have a small total number of users in our system, we recommend [GW09, $\mathcal{S}$3.1]. But if the value of N is very large, then [Wat09] will perform equally well during decryption. And because the adaptive security provided by [Wat09] is stronger than the semi-static security provided by [GW09, $\mathcal{S}$3.1], we especially recommend [Wat09] when the total number of system users $N \geq 1M$.

If the primary goal of our broadcast system is to achieve efficient encryption times, then we recommend any pairing-based system *except* [GW09, $\mathcal{S}$3.2]. Then, depending on the desired level of security, we would choose either the statically secure [BGW05, $\mathcal{S}$3.1], the semi-statically secure [GW09, $\mathcal{S}$3.1], or the adaptively secure [Wat09] broadcast system.

3.6.6 For Further Theoretical Analysis

Now, based on our runtime analysis, we count group operations and construct general big-O notation to represent the time taken for each of these systems to run. We use our experimental results to help construct our theoretical runtime models. For the pairing-based schemes, we will denote λ_1 and λ_2 as a single group multiplication operation over $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. Exponentiations will be denoted by ${\lambda_1}^3$ and ${\lambda_2}^3$. We will let e denote a single pairing operation. For the baseline scheme, we will simply use λ_0 and ${\lambda_0}^3$ to represent a single multiplication and exponentiation over the elliptic curve group $\mathbb{G}$, respectively. In our big-O notation construction, note that we assume the time taken for a single group multiplication is $O(\lambda)$ and the time for a single exponentiation is $O(\lambda^3)$. As an example, if we write a big-O notation as $O(X + \lambda \cdot Y)$, then we mean that the runtime for this algorithm is dominated by $O(Y)$ group multiplications (over $\mathbb{G}_1$ or $\mathbb{G}_2$) and X miscellaneous $O(1)$ operations that individually take much less time than single group multiplications or exponentiations.

In Table 10, there are a few operations that we did not count. We did not count multiplications or exponentiations over $\mathbb{G}_T$ because they did not significantly impact runtimes in any of the schemes. We also did not count any addition/subtraction operations over $\mathbb{Z}_p$ because they were only used to compute $s = s_1 + s_2$ and $r = r_1 + r_2$ in the [Wat09] broadcast system. Additionally, runtimes for the setup phase for [Wat09] and [GW09, $\mathcal{S}$3.1] were dominated by choosing N random generators $\in \mathbb{G}_1$. Since we did not define a symbol for choosing a random generator as an "operation", this is not shown in Table 10. When we use big-O notation to describe the time that the setup phase took for these two schemes (see Table 11), we use δ_1 to denote the time required to choose a single random generator in $\mathbb{G}_1$. Other than for these two setup functions, all of the time complexities for the schemes can easily be derived from Table 10. The big-O notation is best to refer to if we want to know which operation(s) are dominating runtimes for a given function in a broadcast system based on the operations that we defined above. It is not helpful when analyzing the differences in the runtimes of each of these individual operations. For that, it is better to refer to the experimental runtimes given in Table 8.

3.7 Comparison of IBBE Systems

We now compare the identity-based broadcast encryption systems from [GW09, $\mathcal{S}$4.1], [GW09, $\mathcal{S}$4.3.1], and [GW19, $\mathcal{S}$3.1]. We implement all of these schemes almost exactly as they are described in the papers, with a couple of changes that we mention below.

3.7.1 A Summary Of The IBBE Systems We Implemented

Here, we give a brief, high-level summary of each system. We refer readers to the papers to understand how each system works in more detail. Our summaries are as follows:

1. The first system we implement is an adaptively secure, "Initial IBBE Construction" from [GW09, $\mathcal{S}$4.1]. We first initialize the system with the total number users in the system and a predetermined, maximal subset size of users, denoted by N and ℓ, respectively. The maximal subset size ℓ should satisfy the condition that $1 < \ell \leq N$. This IBBE system does not work when $\ell = 1$ because it relies on the computation of $\ell - 2$ degree polynomials during decryption, and a polynomial of a negative degree would not work. In our system, the public key size increases linearly with ℓ and the private key size is contant. Even though the size of the broadcast ciphertext Hdr is technically constant, the ciphertext also includes a long tag τ. If we consider the $\ell - 1$ degree polynomial τ a part of the ciphertext, then we see that ciphertext size also increases linearly with ℓ. In addition to the Setup, Enc, KeyGen, Dec, we had an additional TagGen function to generate the $\ell - 1$ degree polynomial τ and a TagEncrypt function to help with encryption.

2. The second system we implement is a simple variant of the first, adaptively secure system. We omit the long tag τ from the ciphertext and simply output $\tau \leftarrow F(x) \leftarrow 1$. Hence, the system has constant size ciphertexts and private keys. However, this also makes the system less secure. This system, described in more detail in [GW09, $\mathcal{S}$4.3.1], is secure against semi-static adversaries. Note that since we do not need to compute the tag τ, this allows us to omit the additional helper functions TagGen and TagEncrypt from our system. Also, in the implementation, we realized that the following terms in the public key PK became unnecessary: $\{g_2^{\alpha^k} : k \in [0, \ell - 2]\}$ and $\{g_1^{\alpha^j} : j \in [2, l]\}$. We still needed to include g_1 and g_1^{α} in the public key in order to compute $C_3 = g_1^{(1 \cdot t)}$ and $C_4 = e(g_1, \hat{g_2})^{\alpha^{\ell-1} \cdot 1 \cdot t}$ as $e(g_1^{\alpha}, (\hat{g_2}^{\alpha^{\ell}}))^{1 \cdot t}$ during encryption.

3. The third system we implement is an adaptively secure IBBE system from [GW19, $\mathcal{S}$3.1]. In this system, public key and private key sizes increase linearly with the total number of system users m, while ciphertext sizes remain constant. This system is presented as a building block in constructing an efficient, revocable IBBE system. Though we do not implement the revocable system, we implement this system and compare it to the two IBBE constructions that we implement from [GW09]. In the paper, the authors use the variable m to denote the total number of users in the broadcast system and the variable n to denote the size of the privileged set of users $S = \{ID_1, ID_2, \ldots, ID_n\}$. However, we use the same nomenclature as we use for the other systems: instead of m, we will use N to denote the total number of system users, and instead of n, we use $|S|$ to denote the size of the privileged set.

3.7.2 Optimizations For Efficiency And Functionality

We make the following changes to each of the systems:

- In [GW09, $\mathcal{S}$4.1] and [GW09, $\mathcal{S}$4.3.1], we precompute $K = e(g_1, \hat{g_2})^{\gamma \cdot \alpha^{\ell-1}}$ in the setup phase, and then raise it to the power of t during encryption.

- In the construction for [GW09, $\mathcal{S}$4.1], the authors define i to denote the index of a user such that $i \in \{1, 2, \ldots, N\}$. However, we recognize that this is an identity-based system, so we replace the integer index i with an $ID \in \mathbb{Z}_p$. This allows the public key of an individual user in the system to reflect some unique information about the identity of the user (e.g. a user's IP address) instead of

simply being denoted by a predefined index. We make the same change when implementing the variant of this system described in [GW09, $\mathcal{S}$4.3.1].

- In the adaptively secure IBBE system described in [GW19, $\mathcal{S}$3.1], the construction given is not the most efficient possible system in the Type-III pairing setting. In order to make our system more efficient, we simply switch g_1 and g_2. For example, in the setup, instead of computing $\mathbf{U}_1 = g_1^{\mathbf{u}}$, we instead compute $\mathbf{U}_1 = g_2^{\mathbf{u}}$, and we repeat this switching of g_1 and g_2 all throughout the system. Not only does this make our system more efficient, but it also reduces the size of the public key and the private key. Now, during key generation, the $O(N)$ multiplications and exponentiations to calculate $K_{4,i}$ and $K_{5,i}$ for $i \in \{1, 2, \ldots, N\}$ are all computed over $\mathbb{G}_1$ instead of over $\mathbb{G}_2$. We recall that a group element in $\mathbb{G}_1$ takes up 32 bytes of space, while an element in $\mathbb{G}_2$ occupies 64 bytes. Now that $K_{4,i}$ and $K_{5,i}$ consist of elements in $\mathbb{G}_1$, the sizes of these private key components decrease by 50%. There is a similar reduction in the size of the public key because the $N+1$ dimensional vectors $g_2^{\mathbf{u}_1}$ and $g_2^{\mathbf{u}_2}$ are reduced to $g_1^{\mathbf{u}_1}$ and $g_1^{\mathbf{u}_2}$. This also reduces setup times. Calculating $g_1^{\mathbf{u}_1} = (g_1^{u_{1,0}}, g_1^{u_{1,1}}, \ldots, g_1^{u_{1,n}})$ requires $N+1$ exponentiations which are faster over $\mathbb{G}_1$ than over $\mathbb{G}_2$. Additionally, during decryption, the $2m-2$ multiplications required to compute $\prod_{i=1}^{m}(K_{4,i})^{y_i}$ and $\prod_{i=1}^{m}(K_{5,i})^{y_i}$ are all done faster over $\mathbb{G}_1$. The only downside to this switch is that encryption will take marginally longer, since the $|S|$ multiplications required to compute C_3 are now computed over $\mathbb{G}_2$ instead of $\mathbb{G}_1$. But the benefits outweigh the downsides, and we see, based on our analysis, that switching the generators g_1 and g_2 gives us, overall, a more efficient IBBE system.

3.7.3 Runtime Comparison: Setup Times

We now compare the IBBE systems to each other. Since the baseline scheme is not an identity-based broadcast encryption scheme, we do not include it in this comparison. In Table 12, we give the runtimes. We recall that for [GW09, $\mathcal{S}$4.1] and [GW09, $\mathcal{S}$4.3.1], the maximal size of the subset of users ℓ was predetermined in the setup function. When testing runtimes for these two schemes, we set the actual size of the privileged set $|S|$ equal to ℓ.

Looking at the setup times, we first notice that for the larger values of N, the setup time for [GW09, $\mathcal{S}$4.3.1] is always roughly half of the setup time for [GW09, $\mathcal{S}$4.1]. For both these schemes, setup times are dominated by the $O(\ell)$ exponentiations needed to compute a set of group elements. This set included in the public key PK. In the case of [GW09, $\mathcal{S}$4.1], that set is $\{g_1^{\alpha^j}, \hat{g_1}^{\alpha^j}, g_2^{\alpha^k}, \hat{g_2}^{\alpha^k} : j \in [0, \ell],\ k \in [0, l-2]\}$. For [GW09, $\mathcal{S}$4.3.1], we omit some terms and calculate the set $\{g_1, g_1^{\alpha}, \hat{g_1}^{\alpha^j}, g_2^{\alpha^k} : j \in [0, \ell], k \in [0, \ell-2]\}$. Note that the size of the set that needs to be computed for [GW09, $\mathcal{S}$4.3.1] is roughly half the size of the set in [GW09, $\mathcal{S}$4.1]. Hence, computing half the public key takes only half the time. However, when N is small, the value ℓ is also small enough that there is a negligible difference in times to compute the two subsets.

We also see that setup for [GW19, $\mathcal{S}$3.1] took much longer than the other two schemes. This can be simply explained by observing that the time that it takes to generate random exponents for two $N+1$ dimensional vectors *and* exponentiate $O(N)$ times over $\mathbb{G}_1$ is much greater than the time it takes to only exponentiate $O(\ell)$ times over $\mathbb{G}_1$ as in [GW09, $\mathcal{S}$4.1] and [GW09, $\mathcal{S}$4.3.1], when ℓ is a mere 10% of N. More specifically, the setup function in [GW19, $\mathcal{S}$3.1] takes longer because of the time to generate random vectors $\mathbf{u}_1$ and $\mathbf{u}_2 \in \mathbb{Z}_p^{N+1}$ and to use those vectors to exponentiate $g_1^{\mathbf{u}_1}$ and $g_1^{\mathbf{u}_2}$.

3.7.4 Encryption Times

On a first glance at the table, it seems that encryption times increase exponentially for all of the schemes. However, this is not the case. We will once again explain encryption times in a series of observations:

- We start by looking at [GW09, $\mathcal{S}$4.3.1]. Encryption times are dominated by computing $P(x) = \prod_{j=1}^{\ell}(x - i_j) = (x-i_1)(x-i_2)\ldots(x-i_\ell)$. From the expansion of this polynomial, we see that computing the product of the first two binomials requires 4 multiplications over $\mathbb{Z}_p$. Then we multiply the resultant trinomial by another binomial, which takes 6 multiplications. Then the next polynomial product will

take a total of 8 multiplications, and so on. Overall, this step takes $2 \cdot (2+3+\cdots+\ell-1)$ multiplications over $\mathbb{Z}_p$. Noting this is an arithmetic progression, this simplifies to $2 \cdot (\frac{(\ell-1)\cdot\ell}{2} - 1) = \ell^2 - \ell - 2 = O(\ell^2)$ multiplications over $\mathbb{Z}_p$. So we see that encryption times are quadratic with respect to the maximal subset size of users in the system. A benefit of this is that encryption times will remain small no matter how large the value of N is as long as the maximal subset size ℓ remains small. However, increasing the value of ℓ past a certain threshold causes encryption times to increase drastically, as we see in the runtimes table.

- Next we analyze [GW09, $\mathcal{S}$4.1]. In addition to the $O(\ell^2)$ exponentiations to compute $P(x)$, we are told to generate a random $\ell - 1$ degree polynomial $\tau = F(x)$ which satisfies the condition:

$$F(n+j) = 1,\ j \in [k+1, \ell], \text{ where } k = |S|.$$

The authors of the paper suggest generating $F(x)$ by interpolating from k random values in $\mathbb{Z}_p$ and $F(n+j) = 1,\ j \in [k+1, \ell]$. If we were to do this, then a Lagrange interpolation polynomial could be used to interpolate $F(x)$ from the points $(x_1, y_1), (x_2, y_2), \ldots, (x_\ell, y_\ell)$, given by

$$F(x) = \sum_{j=1}^{\ell} y_j \cdot \varphi_j(x), \quad \varphi_j(x) = \prod_{\substack{m=1 \\ m \neq j}}^{\ell} \left(\frac{x - x_m}{x_j - x_m} \right)$$

On a first glance, it might seem that this algorithm requires $O(\ell^3)$ multiplications over the finite field Z_p. Computing the numerator of $\varphi_j(x)$ as $(x-x_1)(x-x_2)\ldots(x-x_{j-1})(x-x_{j+1})\ldots(x-x_\ell)$ requires $O(\ell^2)$ multiplications over Z_p, just like computing $P(x)$. Since $\varphi_j(x)$ has to be computed for every $j \in \{1, 2, \ldots, \ell\}$, this interpolation would run in cubic time. But there is a faster way to do this. First, we precompute the polynomial $G(x) = \prod_{m=1}^{\ell}(x - x_m)$. Then for each $j \in \{1, 2, \ldots, \ell\}$, we use synthetic division to calculate the numerator for $\varphi_j(x)$ as $T(x) = \frac{G(x)}{x - x_j}$. Then we simply use Horner's method for polynomial evaluation to evaluate the denominator of $\varphi_j(x)$ as $T(x_j)$. Finally, we divide the numerator by the denominator. Precomputation of the polynomial $G(x)$ takes $O(\ell^2)$ time, but it is only done once. For all $j \in \{1, 2, \ldots, \ell\}$, $\varphi_j(x)$ is now computed in linear time. Hence, this reduces the overall time complexity of the Lagrange Interpolation from $O(\ell^3)$ to $O(\ell^2)$.

But it turns out that we do not even need to use a Lagrange Interpolation. We can simply calculate the polynomial as:

$$F(x) = (\textbf{random k - 1 degree polynomial}) \cdot \Big(\prod_{j=k+1}^{\ell} (x - (n+j)) \Big) + 1$$

Note that our implementation tests runtimes by setting $k = \ell = |S|$. So the time to generate $F(x)$ is dominated by the time to generate $O(|S|)$ random coefficients over $\mathbb{Z}_p$ for the random $k - 1 = |S| - 1$ degree polynomial. This linear-time algorithm is much faster than the alternate quadratic-time LaGrange interpolation algorithm.

- Encryption times for [GW19, $\mathcal{S}$3.1] appear to be quite similar to the times for [GW09, $\mathcal{S}$4.3.1]. In [GW19, $\mathcal{S}$3.1], encryption times are dominated by computing a polynomial $P_S[Z] = \prod_{ID_j \in S} (Z - ID_j)$. This is very similar to the polynomial $P(x)$ from [GW09, $\mathcal{S}$4.3.1]. The only difference is that the time that it takes to compute this polynomial increases with the square of the size of the subset of users $|S|$, while in [GW09, $\mathcal{S}$4.3.1] the time to compute $P(x)$ increases with the square of the maximal subset size ℓ. But since we set $|S| = \ell$ when testing runtimes, this does not cause a significant difference in encryption times.

3.7.5 Key Generation Times

Looking at key generation times, we see that the times for [GW09, $\mathcal{S}$4.1] and [GW09, $\mathcal{S}$4.3.1] appear to remain constant as N increases. This makes sense, because private key sizes for both of these schemes are constant. The key generation for a single user relies on an efficient, pseudo-random function, which is independent of the total number of users in the system. On the other hand, in [GW19, $\mathcal{S}$3.1], the private key for a single user has size $O(\lambda \cdot N)$. Key generation runtimes are determined by two significant steps. First, we choose N random $ktag_1, ktag_2, \ldots, ktag_N \in \mathbb{Z}_p$. Then, while calculating $K_{4,i}$ and $K_{5,i}$ for $i \in \{1, 2, \ldots, N\}$, we evaluate $O(N)$ group exponentiations and multiplications over $\mathbb{G}_1$. This explains why key generation times for [GW19, $\mathcal{S}$3.1] appear to scale linearly as N increases.

3.7.6 Decryption Times

Decryption times are directly analogous to encryption times. For [GW09, $\mathcal{S}$4.3.1], decryption times are essentially the same as encryption times because decryption time is dominated by the time required to compute $P(x) = (x-i_1)(x-i_2)\ldots(x-i_\ell)$. This exact same polynomial is computed during encryption. This also happens to be the case for [GW19, $\mathcal{S}$3.1]: decryption time is dominated by computing the polynomial $P_S[Z] = \prod_{ID_j \in S}(Z - ID_j)$, the exact same polynomial which dominates encryption times for this scheme. As for [GW09, $\mathcal{S}$4.1], decryption consistently takes slightly longer than decryption for [GW09, $\mathcal{S}$4.3.1], In both of these schemes, decryption requires computing the polynomial $P(x)$. But for [GW09, $\mathcal{S}$4.1], we also need to use synethetic division to compute:

$$F_i(x) = \frac{F(x) - F(i)}{x - i}$$

Since $F(x)$ is an $\ell - 1$ degree polynomial, this synthetic division step requires $O(\ell)$ additions and multiplications over $\mathbb{Z}_p$. This explains the slightly longer decryption times for [GW09, $\mathcal{S}$4.1] compared to [GW09, $\mathcal{S}$4.3.1].

3.7.7 Overall Evaluation

Based on the runtime evaluation in Table 12 and the size evaluation in Table 13, if we had to recommend one of these IBBE systems over the others, it would be [GW09, $\mathcal{S}$4.3.1]. The ciphertext and private key sizes are constant, and the public key size is the smallest out of all the systems. As far as runtimes go, [GW09, $\mathcal{S}$4.3.1] has the fastest setup times and constant key generation times. Encryption times are slightly faster than those of [GW09, $\mathcal{S}$4.1] and equally as fast as those of [GW19, $\mathcal{S}$3.1], while decryption times are approximately as fast as the other schemes. The only disadvantage of this IBBE system is that it is secure against weaker semi-static adversaries, rather than adaptive adversaries.

If we need an adaptively secure IBBE system, then we can choose between [GW09, $\mathcal{S}$4.1] and [GW19, $\mathcal{S}$3.1]. We have two situations. First is the case where the number of privileged users $|S|$ is very small in comparison to N. This works well with [GW09, $\mathcal{S}$4.1] because it allows for relatively efficient encryption while keeping the public key and ciphertext sizes small. Recall that the public key and ciphertext sizes for this scheme are $O(\lambda \cdot \ell)$, and encryption times are dominated by $O(\ell^2)$ multiplications over $\mathbb{Z}_p$. So when $\ell << N$, we are able to keep both ℓ and ℓ^2 small, achieving short public key and ciphertext sizes and fast encryption times. We acknowledge that despite the fast encryption times we would achieve, the encryption times for [GW19, $\mathcal{S}$3.1] would still be faster. But the downside to choosing [GW19, $\mathcal{S}$3.1] over [GW09, $\mathcal{S}$4.1] in this case is that the public key and private keys would be unnecessarily large, with size of $O(\lambda \cdot N)$. In the second case, the number of privileged users $|S|$ in the system is a large fraction of the total number of users N. In this case, we recommend [GW19, $\mathcal{S}$3.1] in order to achieve efficient encryption. The only downside we see to this is that it would take a very long time to generate the keys for the large number of privileged users in the system. Still, if the primary goal is efficient encryption, and the resources are available to handle such a situation, [GW19, $\mathcal{S}$3.1] would be the better IBBE system when $|S|$ is very large.

Comparing the IBBE systems to the general public key broadcast encryption systems from 3.6, we start by pointing out that [GW19, $\mathcal{S}$3.1] is similar in many ways to [Wat09]. Both these systems are adaptively

secure, have constant size ciphertexts, and public key and private key sizes that grow linearly with the total number of users in the broadcast system. If we compare the runtimes, though, we see that setup, key generation, encryption, and decryption times are all slower for [GW19, $\mathcal{S}$3.1]. Encryption and decryption times also grow much faster for [GW19, $\mathcal{S}$3.1] as N increases. This is due to a general relationship that holds for all of the public key broadcast systems in comparison to the IBBE systems: encryption and decryption times increase linearly for public key broadcast systems, but they increase quadratically for IBBE systems. Just based on this, we would always recommend a public key broadcast encryption system over an IBBE system if the identity-based functionality is not needed in the system.

3.8 Comparison of Broadcast Tracing Systems

We compare a wide range of tracing systems. Among the systems that we compare, we have a private-linear broadcast encryption (PLBE) system from [GKSW09], an augmented broadcast encryption (ABBE) system from [GKSW10], and a risky broadcast and trace multi-scheme from [Zha20].

First we point out that the PLBE system and the ABBE system constructions are from the same paper. However, the authors chose to eliminate their PLBE construction from the 2010 version of their paper. So [GKSW09] refers to the 2009 version of their paper which contains the PLBE construction.

3.8.1 Efficiency Optimizations for [GKSW09]

Next, we point out that the PLBE and ABBE system constructions given in [GKSW09] and [GKSW10] are based on the constructions originally given in [BSW06] and [BW06], respectively. But the original constructions were written in the composite order pairing setting. Luckily, the authors of [GKSW10] presented a construction of [BW06] using asymmetric bilinear groups. This is the version that we implement – specifically, the construction given in [GKSW10, $\mathcal{S}$5.2]. It is slightly different from the original composite order construction. In the context of our implementation, though, this construction is not optimized for efficiency. A more efficient version of the construction is to switch certain elements in $\mathbb{G}_1$ and $\mathbb{G}_2$. We make a few changes to the described scheme:

1. In $Setup_{AugBE}$, $u_1, u_2, \ldots, u_m \in \mathbb{G}_1$ instead of $u_1, u_2, \ldots, u_m \in \mathbb{G}_2$. This reduces the time required to compute the public key, since it takes much less time to choose a random element $u_i \in \mathbb{G}_1$ as compared to $u_i \in \mathbb{G}_2$ (see Table 2). Additionally, it reduces the size of the public key, which includes these u_i elements. Recall that an element in $\mathbb{G}_1$ occupies 32 bytes of space, while an element in $\mathbb{G}_2$ occupies 64 bytes of space.

2. Calculate the key for a single user in the system as $K_{(x,y)} = \{g_1^{\alpha_x} \cdot g_1^{r_x c_y} \cdot u_y^{\delta_{x,y}}, g_2^{\delta_{x,y}}, \forall i, (i \neq y), u_y^{\delta_{x,y}}\}$. We essentially just placed the group elements in the secret key which were originally in $\mathbb{G}_1$ in $\mathbb{G}_2$, and vice versa. We see that this is the optimal configuration. Since all elements $u_y \in \mathbb{G}_1$ instead of in $\mathbb{G}_2$, private key sizes are smaller. It also takes less time to compute $K_{(x,y)}$ in this modified construction, because the $N-1$ exponentiations required to calculate $u_y^{\delta_{(x,y)}}, \forall i, (i \neq y)$ are computed more efficiently over $\mathbb{G}_1$. As a final note, encryption times will also be more efficient due to this switch. Encryption times are dominated by the time to compute the inner product in T_x as $\prod_{k \in S_x} u_k$. The $|S| - 1$ group multiplications to calculate this product are much faster with $u_k \in \mathbb{G}_1$ as oppose to $u_k \in \mathbb{G}_2$.

3. During encryption, $A_x \in \mathbb{G}_2$. For example, if $x < i$, then calculate $A_x = g_2^{a_x}$ instead of $A_x = g_1^{a_x}$. This allows us to compute the pairing $e(K'_{x,y}, A_x)$ during decryption.

3.8.2 Converting ABBE to PLBE and Optimizing for Efficiency

Unfortunately, the authors of [GKSW09] only describe the PLBE system in the symmetric pairing setting. We recall that a PLBE system is merely a simpler version of an ABBE system. So all that really needs to be done to obtain a PLBE system is eliminate some components of the ABBE system. We describe the changes

that we make below. Note that these changes are made after modifying the ABBE system as we described above.

1. $Setup_{PLBE}$: Eliminate $u_1, u_2, u_m \in \mathbb{G}_2$ from the public key. The secret key for each user is $K_{(x,y)} = g_2^{\alpha_x} \cdot g_2^{r_x c_y} \cdot u_y^{\delta_{x,y}} \in \mathbb{G}_2$.

2. Enc_{PLBE} : As in the original ABBE system, $A_x \in \mathbb{G}_1$. Since $K_{(x,y)} \in \mathbb{G}_2$, this later allows us to calculate $e(A_x, K_{(x,y)})$ in decryption. Also, because $u_1, u_2, \ldots u_m$ was eliminated from the public key, T_x no longer has to be calculated or included in the ciphertext.

3. Dec_{PLBE} : Output $M = (B_x \cdot e(R_x, C_y)) \ / \ (e(A_x, K_{(x,y)}) \cdot e(\tilde{R}_x, \tilde{C}_y))$.

By simplifying this ABBE system to a PLBE system, we are reducing the private key size from $O(\lambda \cdot \sqrt{N})$ to constant size private keys of $O(\lambda)$. The disadvantage of this simpler scheme is that the master authority is responsible for finding a secure way to distribute the N private keys to each of the individual users outside of the system. This is unlike the ABBE system in which private keys are delegated to users within the system, which requires blowing up the private key size.

3.8.3 An Additional Implementation-Specific Modification

Once we started testing runtimes for these schemes, we noticed a major problem. Runtimes could not be tested for $N = 100$K or $N = 1$M for the ABBE system because the computer on which we were running the program ran out of memory during the $Setup_{AugBE}$ phase. We refer readers to [GKSW10, $\mathcal{S}$5.2] to see that in the setup phase, the computer would need to generate and store N private keys $K_{(x,y)}$, each of size $O(\lambda \cdot N)$. In total, $O(N^2)$ elements would need to be stored in memory. This was clearly too much. To rectify this issue, we modified each of the tracing systems in a way that would allow us to test runtimes. Instead of generating the key for all N users in the setup function, we introduce a KeyGen function which generates the key for a single user from a master secret key MSK which we define during the setup phase. For the ABBE system, we make the following changes:

1. In $Setup_{AugBE}$, calculate PK as described. Define $\text{MSK} = \{PK, r_1, r_2, \ldots, r_m, \alpha_1, \alpha_2, \ldots, \alpha_m, c_1, c_2, \ldots, c_m\}$. Output (PK, MSK).

2. Introduce a new function $KeyGen_{AugBE}$, which takes as input the master secret key MSK and a user (x, y). Output $K_{(x,y)}$.

We make this simple change for all of the tracing systems that we implement. For the PLBE system, MSK is the same as for the ABBE system, and $K_{(x,y)}$ is calculated as we described it in 1. For the risky broadcast and trace scheme described in [Zha20, $\mathcal{S}$9.3], a master secret key MSK is included in the tracing key generated during the setup phase. The tracing key includes MSK along with additional parameters that need to be known to generate the key for an individual user in the system. The $KeyGen$ function takes as input this tracing key tk and a user (j, i), and outputs the key for that user.

3.8.4 MTB Scheme Efficiency Optimization

Finally, we make slight changes to the Mixed Bit Matching with Threshold Broadcast (MTB) scheme given in [Zha20, $\mathcal{S}$9.2]. This MTB scheme forms the basis of the risky and nonrisky broadcast and trace schemes that we implement from [Zha20, $\mathcal{S}$9.3]. The change is simple: switch *all* of the elements in $\mathbb{G}_1$ and $\mathbb{G}_2$. This reduces the public key size and setup times. As expected, the $O(v)$ exponentiations to compute $g_2^{(\gamma^j, \mathbf{0}, 0) \cdot R^T}$ for $j \in [0, v]$ will perform much faster when these elements are in $\mathbb{G}_1$. And since a single group element in $\mathbb{G}_1$ takes up half as much space as a single group element in $\mathbb{G}_2$, this also reduces the public key size. Encryption times and decryption times remain roughly the same. We explain why this is the case in our encryption and decryption runtime analysis for the tracing schemes. The only disadvantage of this modification is that it

takes slightly longer to generate the key for a single user in the system. The difference is not noticeable when generating the key for just a single user in the system, but if we need to generate the keys for all N users in the broadcast and trace system as efficiently as possible, then we might want to implement the scheme as it is written. But since we only generate the key for a single user while testing runtimes, we make this change.

3.8.5 Initial Runtime Comparison

Now we analyze the runtimes of the tracing systems. In our runtime table, we exclude the nonrisky broadcast and trace system which is briefly described in [Zha20, $\mathcal{S}$9.3]. This scheme was not thoroughly described in [Zha20], so we analyze it separately in more detail in 3.8.10. In this section, whenever we use the reference [Zha20, $\mathcal{S}$9.3], we refer to the risky broadcast and trace scheme given by **Construction 8** in [Zha20].

We point out that the risky broadcast and trace scheme given in [Zha20, $\mathcal{S}$9.3] is a broadcast multi-scheme. Basically, this system sets up M instances of an N-user broadcast system running in parallel. The advantage of this is that all of the instances share the same public key. Encryption and decryption happens to a single instance. In our case, we set $M = 1$ because we only want to evaluate the performance of a single broadcast encryption system. Setup times for this scheme will represent the time taken to set up a single N-user instance of the risky broadcast multi-scheme.

3.8.6 Setup Times

Looking at setup times, we notice that times for [GKSW09] (the PLBE system) appear to be consistently lower than times for [GKSW10] (the ABBE system). This is not surprising. During setup for both of these schemes, runtimes are dominated by computing the public key PK. However, in the PLBE system PK does not need to include $u_1, u_2, \ldots, u_m$. So the difference in runtimes for these two schemes can be explained by the additional time required to choose random $u_1, u_2, \ldots, u_m \in \mathbb{G}_1$ for the ABBE system. Also, since $m := \left\lceil \sqrt{N} \right\rceil$, calculating the public key requires only $O(\lambda \cdot \sqrt{N})$ operations for these two schemes. In contrast, calculating the public key in the setup function for [Zha20, $\mathcal{S}$9.3] requires $O(\lambda \cdot N)$ exponentiations to calculate $\left(g_1{}^{(\gamma^j,\, \mathbf{0},\, 0)\cdot R^T}{}_{j\in[0,v]}\right)$, since $v = N$. This explains why setup times appear to increase linearly for [Zha20, $\mathcal{S}$9.3] and increase sub-linearly for [GKSW09] and [GKSW10].

3.8.7 Key Generation Times

Key generation times for each of the schemes are best explained as a series of observations:

- The $KeyGen_{PLBE}$ function in [GKSW09] generates a constant size private key for a single user in the system. As expected, this takes a constant amount of time roughly equal to the amount of time to perform the three exponentiations and two multiplications over $\mathbb{G}_2$ in calculating $K_{(x,y)} = g_2^{\alpha_x} \cdot g_2^{r_x c_y} \cdot u_y^{\delta_{x,y}} \in \mathbb{G}_2$.
- The $KeyGen_{AugBE}$ function in [GKSW10] generates a private key of size $O(\lambda \cdot \sqrt{N})$ for a single user in the system. as $K_{(x,y)} = \{g_1^{\alpha_x} \cdot g_1^{r_x c_y} \cdot u_y^{\delta_{x,y}}, g_2^{\delta_{x,y}}, \forall i, (i \neq y), u_y^{\delta_{x,y}}\}$. Looking more closely, $\forall i, (i \neq y)$ refers to all $i \in \{1, 2, \ldots, \sqrt{N}\}$. This means that calculating the private key for a single user takes $O(\sqrt{N})$ exponentiations over $\mathbb{G}_1$ with $\forall i, (i \neq y), u_y^{\delta_{x,y}}$. Indeed, Table 14 shows that as N increases for [GKSW10], the time taken to compute the private key for a single user increases sub linearly.
- Key generation times for [Zha20, $\mathcal{S}$9.3] remain roughly constant. Calculating the key for a user u with attribute $\mathbf{x}$ as $sk = \left(g_2^{(\tau_\theta/(1-\gamma/u),\, \mathbf{x}\cdot D_\ell,\, \eta_\ell)\cdot R^{-1}}\right)$ requires, in addition to $n = 4$ exponentiations over $\mathbb{G}_2$, the transformation of two vectors in the exponent by 4 dimensional square matrices. These operations are slightly more costly in comparison to the three exponentiations and two multiplications over $\mathbb{G}_2$ to calculate the key for a single user in [GKSW09]. This explains why key generation times are slightly longer for this scheme.

For all these schemes, with $N <= 1M$, individual user key generation times remain under a tenth of a second. This is quite efficient. A slight disadvantage is that if necessary, generating the keys for all N users in the system in [GKSW10] would take much longer than for either the PLBE system ([GKSW09]) or the risky broadcast and trace system ([Zha20, $\mathcal{S}$9.3]). This is as expected. As we have mentioned already, PLBE is a simpler version of this ABBE system. Constant key generation times are one of the advantages that come from this simplified system. The same is true of the risky broadcast and trace system, but the disadvantage of this system is that because of the constant size private keys, it is "risky". In 3.8.10, we see that removing the risk from the risky broadcast and trace system requires expanding the secret key size to $O(\lambda \cdot N)$ using a risk mitigation compiler described in [Zha20, $\mathcal{S}$2.3].

3.8.8 Encryption Times

Encryption times for [GKSW09] (PLBE) and [GKSW10] (ABBE) are very similar. For both of these schemes, encryption means generating the ciphertext for all N users (x, y) in the tracing system. Generating the ciphertext for a single user involves a constant number of exponentiations over $\mathbb{G}_1$ and $\mathbb{G}_2$ for the PLBE system.

For the ABBE system, generating the ciphertext for a single user means additionally computing $T_x = (\prod_{k \in S_x} u_k)^{\mathbf{exp}}$, where the value of **exp** depends on how the x coordinate of a certain user compares to the x-index i to which encryption occurs. If j is the y-index to which encryption occurs, this ensures that users with $x < i$ cannot decrypt the ciphertext, users with $x > i$ can decrypt, and users with $x = i$ and $y >= j$ can decrypt. In the expression for T_x, S_x is a set which contains the unique y coordinates of users in the subset S to which encryption occurs. The amount of time to compute this set is very small because we use hash sets for efficient computation. But the size of the set $|S_x|$ varies depending on how many unique y-coordinates are among the subset of users. This is significant because the time that it takes to compute the inner product in T_x depends on $|S_x|$, requiring $|S_x| - 1$ group multiplications over $\mathbb{G}_1$. As N increases, $|S|$ increases. With more users in S, there is a greater probability of obtaining a higher number of users with unique y-values, and therefore a greater chance of obtaining a larger $|S_x|$. So we expect that as N increases, the average amount of time to generate the ciphertext for a single individual user also increases. For both this ABBE system and the PLBE system, encryption means generating ciphertext for all N users in the system. This means that as N increases, the discrepancy between the amount of time to encrypt to the ABBE system and the amount of time to encrypt to the PLBE system will increase. Indeed, if we look at the Table 14, this appears to be the case.

Encryption for [Zha20, $\mathcal{S}$9.3] is a complicated, multi-step process. Runtimes are dominated by computing the second half of the ciphertext $c_2 = g_1^{(\alpha \prod_{s \in S} (1-\gamma/s), \mathbf{0}, 0) \cdot R^T)}$. For the benefit of the reader, we choose to explain it in a series of steps.

1. First calculate the coefficients of the polynomial $A(\gamma) = \prod_{s \in S} (1 - \gamma/s)$. Multiplying out these $|S| - 1$ binomials takes quadratic time. If we multiply two binomials, it will require 4 multiplications. Then multiplying the resultant trinomial by another binomial will take 6 multiplications. We continue this process to get a total of:

$$2 \cdot (2 + 3 + \cdots + |S| - 1) \;=\; 2 \cdot \left(\frac{(|S| - 1)(|S|)}{2} - 1 \right) \;=\; |S|^2 - |S| \; -2 \text{ total multiplications over } \mathbb{Z}_p$$

   Overall, this step requires $O(|S|^2)$ multiplications over $\mathbb{Z}_p$.

2. Use the coefficients of the polynomial from the first step and the components of the public key to calculate $g_1^{(\prod_{s \in S} (1-\gamma/s), \mathbf{0}, 0) \cdot R^T)}$. First, match a vector in PK to its corresponding coefficient. Then exponentiate each element in the vector by that coefficient. Repeat for all $\left(g_2^{(\gamma^j, \mathbf{0}, 0) \cdot R^T)} \right)_{j \in [0, v]} \in PK$. Finally, multiply together the corresponding elements of all N 4-dimensional vectors. This process takes a total of $4N$ exponentiations and $4(N - 1)$ multiplications over $\mathbb{G}_1$.

3. To finally calculate $c_2 = g_1^{(\alpha \prod_{s \in S} (1-\gamma/s), \mathbf{0}, 0) \cdot R^T)}$, multiply each element in the vector result from the previous step by α. Since the risky broadcast and trace system deals with 4 dimensional vectors, this only takes a constant 4 multiplications over $\mathbb{G}_1$.

Out of all of these steps, it is the $O(|S|^2)$ multiplications over $\mathbb{Z}_p$ which dominate runtimes. As we mentioned in 3.8.4, the only change that we made when implementing this scheme was that we switched the elements in $\mathbb{G}_1$ and $\mathbb{G}_2$. This had a negligible effect on encryption times because even though the $4N$ exponentiations and $4(N-1)$ multiplications over $\mathbb{G}_2$ are now more efficiently computed over $\mathbb{G}_1$, these operations are still negligible in comparison to the $|S|^2 - |S| - 2$ over $\mathbb{Z}_p$. Just like for the IBBE systems in 3.7, encryption times for the risky broadcast and trace system given in [Zha20, $\mathcal{S}$9.3] increase quadratically.

3.8.9 Decryption Times

Finally, decryption times for [GKSW09] and [GKSW10] remain relatively small as N increases, while those for [Zha20, $\mathcal{S}$9.3] increase quadratically.

- For [GKSW09], we compute $M = (B_x \cdot e(R_x, C_y)) \ / \ (e(A_x, K_{(x,y)}) \cdot e(\tilde{R}_x, \tilde{C}_y))$ during decryption. Decryption times remain roughly constant because the decryption process always requires exactly three pairings. For [GKSW10], the decryptor needs to additionally compute $K'_{(x,y)} = g_1^{\alpha_x} \cdot g_1^{r_x c_y} \cdot \prod_{\kappa \in S_x} u_k^{\delta_{x,y}}$ from the components of $K_{(x,y)}$. Remember that S_x is the subset of unique user y coordinates in the subset S. In the "best" case, this set will have size $O(|S| \ / \sqrt{N})$, and in the "worst" case, size $O(\sqrt{N})$ assuming $|S| > \sqrt{N}$. The most probable case will lie somewhere between the best case and the worst case. Though the time cost to compute $K'_{(x,y)}$ depends on the size of S_x, we see that as N and $|S|$ increase, the best case and worst case bounds will shift upwards, increasing the probability of obtaining a larger subset S_x. And because decryption requires $|S_x| + 1$ multiplications to compute $K'_{(x,y)}$, this will result in a higher probability of larger decryption times.

- For [Zha20, $\mathcal{S}$9.3], decryption times are very similar to encryption times. To decrypt for a single user $u \in U$, we have to compute the polynomial $Q(\gamma) = \prod_{j \in S \setminus U} (1 - \gamma/j)$. Just like the polynomial in encryption, this polynomial will require $O(|S|^2)$ multiplications over $\mathbb{Z}_p$. Computing $J_\theta P$ from the components of the universal helper key will take $O(N)$ exponentiations and multiplications over $\mathbb{G}_1$. This process also requires approximately the same number of operations as computing c_2 from the components of PK during encryption. As such, decryption times are very close to encryption times.

Looking at Table 14, encryption times for [Zha20, $\mathcal{S}$9.3] appear to be less than, or roughly equal to those for [GKSW10] when $N <= 1K$. After that, encryption times grow incredibly large at a rapid rate. We might mistakenly come to the conclusion that, like with a few of the IBBE systems, for $|S| << N$, [Zha20, $\mathcal{S}$9.3] would be a good alternative to [GKSW10]. But remember that the second step in encryption for [Zha20, $\mathcal{S}$9.3] requires $O(N)$ multiplications and exponentiations over $\mathbb{G}_1$. So if we had a situation such as $N = 1M$ and $|S| = 1K$, [GKSW10] would still be faster. The only case in which the risky broadcast and trace system has faster encryption times than the ABBE system is for extremely small values of N and $|S|$. Even for these small values, though, decryption times are worse than those for [GKSW10]. So we end by saying that this risky broadcast and trace scheme likely has no practical applicability.

3.8.10 Zhandry's Efficient Broadcast Multischeme

There is a clever way to modify the risky broadcast and trace multischeme given in Section [Zha20, $\mathcal{S}$9.3] to obtain a nonrisky broadcast and trace scheme with variable size public, private, and ciphertext sizes. In particular, the public key, private key, and ciphertext will have sizes $(O(\lambda \cdot N^{1-a}), O(\lambda \cdot N^{1-a}), O(\lambda \cdot N^a))$, respectively, for any $a \in [0, 1]$. In 3.8.5, we showed that the risky broadcast and trace multischeme performs very poorly in comparison to the other traitor tracing schemes because encryption and decryption times increase quadratically with the size of the subset of privileged users in the broadcast system. So, on a first

glance, it might seem pointless to even attempt to implement this modification when runtimes for the original scheme perform so badly.

Two modifications need to be made. First, remove the risk from the risky scheme using a risk mitigation compiler described in [Zha20, $\mathcal{S}$2.3]. Then, apply a user expansion compiler described in [Zha20, $\mathcal{S}$5] to obtain a scheme with variable size public, private, and ciphertext sizes. This process is explained in [Zha20, $\mathcal{S}$9.3], but the author does not give a detailed construction of the scheme. We give a full, detailed construction below:

Setup$(N,\ M,\ T,\ \lambda)$*:* The setup algorithm takes as input the number of users N in the broadcast system, the number of instances M of this system, and a parameter $T = \lfloor N^a \rfloor$. Let $p = \lceil \frac{N}{T} \rceil$ and $q = M \cdot T$. Then, let $n = 2$, $v = p$, $u = 1$, and $t = 1$, and run $\mathsf{Gen}_{\mathsf{MTB}}(u,\ v,\ t,\ n,\ \lambda)$ to get $(PK,\ MSK)$. For each $j \in \{1, 2, \ldots, p \cdot q\}$, choose a random $i_j{}^* \in \{1, 2, \ldots, p\}$. The tracing key is $\mathsf{tk} = MSK,\ (i_j{}^*)_{j \in \{1,2,\ldots,p \cdot q\}}$. Output $(PK,\ \mathsf{tk})$.

KeyGen$((j,\ i),\ \mathsf{tk})$*:* The secret key for a single user $(j,\ i)$ will include the p keys for that user across all independent instances of this broadcast multischeme. For each $X = 1, 2, \ldots, p$, let $j'_X = j + M \cdot T \cdot (X - 1)$. Let $\mathbf{w}_{j'_X,\, i} \in \{0,1\}^2$ be (1, 1), (1, 0), (0, 0) for $i < i^*_{(j'_X)}$, $i = i^*_{(j'_X)}$, and $i > i^*_{(j'_X)}$, respectively. Let $U = \{u'_X\}$, where $u'_X = (j'_X - 1) \cdot p + i$. The secret key for this instance is $\mathsf{sk}_{(j,\, i),\, X} = \mathsf{Extract}_{\mathsf{MTB}}(MSK,\ U,\ \mathbf{w}_{j'_X,\, i})$. Output $\forall X,\ \mathsf{sk}_{(j,\, i),\, X}$.

Encrypt$(PK,\ k,\ S)$*:* To encrypt to the kth instance of the broadcast scheme and a subset of users S, first choose a random instance $X \in \{1, 2, \ldots, p\}$ to encrypt to. Then, partition the subset S into T subsets as follows:

1. Initialize $\hat{S}_x = \{\}$ for $x \in \{1, 2, \ldots, T\}$.
2. For each user $(j, i) \in S$, add i to $\hat{S}_j$.

Let $j_{X_0} = k \cdot T - (T - 1) + M \cdot T \cdot (X - 1)$. Then for each $\ell \in \{j_{X_0}, j_{X_0} + 1, \ldots j_{X_0} + T\}$, define

$$T_{\ell,\, S} = \{\, (\ell - 1) \cdot p + i \quad | \quad i \in \hat{S}_{(\ell - 1) \pmod T} \,\}$$

The ℓth ciphertext component is $(c'_\ell,\ k'_\ell) = \mathsf{Enc}_{\mathsf{MTB}}(PK, T_\ell S)$. Output X, $(c'_{(j_{X_0})}, k'_{(j_{X_0})})$, $(c'_{(j_{X_0}+1)}, k'_{(j_{X_0}+1)})$, $\ldots$, $(c'_{(j_{X_0}+T)}, k'_{(j_{X_0}+T)})$.

Decrypt$((j,\ i),\ \mathsf{sk}_{(j,\, i),\, X},\ h_\theta,\ S,\ c')$*:* Let $j'_X = j + M \cdot T \cdot (X - 1)$. To decrypt the ciphertext for a user $(j,\ i)$, first the ciphertext component corresponding to this user needs to be determined. The corresponding component is c'_ℓ, where $\ell = j'_X + T$ if $j \pmod T$ equals 0, otherwise $\ell = j'_X + j \pmod T$. Define the subset $\hat{S}_\ell$ as follows:

$$\hat{S}_\ell := S \cap \{((j-1) \cdot p + 1) \pmod N,\ ((j-1) \cdot p + 2) \pmod N, \ldots,\ ((j-1) \cdot p + p) \pmod N\}$$

Set $U = \{u'_X\}$, where $u'_X = (j'_X - 1) \cdot p + i$. Then calculate

$$T_{\ell,\, S} = \{\, \forall\, (\hat{j},\ \hat{i}) \in \hat{S}_\ell,\ (j'_X - 1) \cdot p + \hat{i}\}$$

Finally, output $k'_\ell = \mathsf{Dec}_{\mathsf{MTB}}((\mathsf{sk}_{(j,\, i),\, X},\ h_\theta),\ T_{\ell,\, S},\ U,\ c'_\ell)$.

Our construction relies on the idea that every user index u in the system can be expressed as a set of coordinates (k, j, i). The coordinates can be extracted from u as follows:

- If $u \pmod N$ equals 0, then $k = \lfloor \frac{u}{N} \rfloor$. Otherwise, $k = \lfloor \frac{u}{N} \rfloor + 1$.
- $j = \left\lceil \frac{u}{p} \right\rceil$.
- If $u \pmod p$ equals 0, then $i = p$. Otherwise, $i = u \pmod p$.

In this coordinate system, user $u = N \cdot (k-1) + p \cdot (j-1) + i$. It is a 1-index based coordinate system, meaning user $u = 1$ has coordinates $(1, 1, 1)$, user $u = 2$ has coordinates $(1, 1, 2)$, etcetera.

In this new broadcast encryption system, setting $T = \lfloor N^a \rfloor$ yields a scheme in which the public key, private key, and ciphertext size are $N^{(1-a)}$, $N^{(1-a)}$, and N^a, respectively. We implement and test this system for different values of a. Our results are given in Table 16. We set $M = 1$ to test the runtimes for a single broadcast encryption system in the multi-scheme. For $a = 1$ and $N = 1$M, we were unable to obtain encryption and decryption times because the computer on which runtimes were being tested ran out of memory.

3.8.11 Broadcast Multischeme: Theoretical Runtimes

To analyze the runtimes for the nonrisky broadcast and trace system given in Table 16, we first determine the theoretical amount of time that it should take for Setup, KeyGen , Enc, and Dec to run, in terms of N, a, and other known constants. We have the following:

- Looking at Setup, there are two steps that determine runtimes. First is exponentiating the $O(p)$ group elements in $\mathbb{G}_1$ during $\mathsf{Gen}_{\mathsf{MTB}}$ to calculate PK. Since $p = \lceil \frac{N}{T} \rceil$ and $T = \lfloor N^a \rfloor$, this step will require $O(N^{1-a})$ exponentiations over $\mathbb{G}_1$. The second step is choosing N random values ${i_j}^*$ in the range $1, 2, \ldots, p$. Since we consider choosing a random integer in a range a "miscellaneous", negligible operation, the overall big-O runtime notation for Setup, based on the brief definitions we give at the end of 3.6, is $O(\lambda^3 \cdot N^{1-a} + N)$.
- For KeyGen, we first note that the function $\mathsf{Extract}_{\mathsf{MTB}}$ runs in constant time. Since the set U for each user is disjoint, the helper key hk_θ does not need to be included in each secret key and we can instead include one universal helper key for all the secret keys in the public parameters. During key generation, runtimes are determined by generating secret keys for the p instances of the broadcast multi-scheme using $\mathsf{Extract}_{\mathsf{MTB}}$. Key extraction for a single user requires only $n = 4$ exponentiations over $\mathbb{G}_1$. In total, this step requires $4p$ exponentiations over $\mathbb{G}_1$, and runs in time $O(\lambda^3 \cdot N^{1-a})$.
- Encryption is a complex, multi-step process. There are three main steps. First is iterating over the elements in S, and adding them to their corresponding S_j subset. Next, calculate $T_{\ell,S}$ for T values of ℓ. Both of these steps take a negligible amount of time. The most time-consuming part of encryption is using $\mathsf{Enc}_{\mathsf{MTB}}(PK, T_{\ell,S})$ to calculate (c'_ℓ, k'_ℓ) for T values of ℓ. Looking back on our analysis from 3.8.5, $\mathsf{Enc}_{\mathsf{MTB}}$ will require $O(|T_{\ell,S}|^2)$ multiplications over $\mathbb{Z}_p$. Due to this, encryption times will be the most efficient when there are a small total number of subsets $T_{\ell,S}$ with a small average number of users per subset. Another way to think about it is that encryption times will be most efficient when the users in S are most spread out over the $T_{\ell,S}$ subsets.
- Decryption times are determined by two steps. First are the $O(p)$ miscellaneous, negligible individual operations required to compute the individual set $T_{\ell,S}$ to decrypt c_ℓ. Then there is the call to $\mathsf{Dec}_{\mathsf{MTB}}((j, i), (\mathsf{sk}_{(j,i),X}, h_\theta), T_{\ell,S}, c'_\ell)$, which is dominated by $O(|T_{\ell,S}|^2)$ multiplications over $\mathbb{Z}_p$. The difference between decryption and encryption is that there is only one call to $\mathsf{Dec}_{\mathsf{MTB}}$ during decryption, while there are T calls to $\mathsf{Enc}_{\mathsf{MTB}}$ during encryption. Overall, decryption times are predominantly dependent on the size of the individual subset $T_{\ell,S}$.

The different Setup and KeyGen times for different values of a given in Table 16 can be explained by the big-O notation that we derived. Encryption and decryption times can be explained conceptually.

3.8.12 Broadcast Multischeme: Encryption And Decryption Times

For encryption times, we first look at the extremes. When $a = 1/3$, there will be a small total number of $T_{\ell,S}$ subsets. Specifically, there will be $T = \lfloor N^a \rfloor$ subsets. However, the issue is that because there are approximately T^3 total users in the broadcast system, there will be a very high average number of users per subset, resulting in poor encryption times. When $a = 1$, we see encryption times that grow linearly with the total number of users in the system. This is as expected, because there will be $T = N$ subsets, each of size $|T_{\ell,S}| = 1$. This is much better than the quadratic encryption times of the risky broadcast and trace scheme. With constant size public keys and private keys, the only disadvantage is that ciphertext sizes grow linearly.

But we can improve encryption times even further. Setting $a = 1/2$ yields the scheme with the fastest encryption times, with a perfect balance between the total number of subsets and the average number of users per subset. Just like the Augmented Broadcast Encryption (ABBE system) from [GKSW10], this scheme has public keys, private keys, and ciphertext sizes of size $O(\sqrt{N})$. For the smaller values of N, encryption times are comparable to those for the ABBE system, but as N increases, the discrepancy in encryption times widens. When $N = 1$M, encryption for [GKSW10] is fifteen times faster than that for this scheme when $a = 1/2$.

Looking at Table 16, we see that when $a = 2/3$, Setup times are faster, KeyGen times are faster, Enc times are slightly slower, and Dec times are faster than when $a = 1/2$. Additionally, setting $a = 2/3$ yields a tracing scheme with a smaller public key and private key of size $N^{1/3}$ and a slightly larger ciphertext of size $N^{2/3}$. Moving from $a = 2/3$ to $a = 3/4$ has a similar effect – Setup times are faster, KeyGen times are faster, and Dec times are slightly faster. The public key and private key are reduced to size $N^{1/4}$, while the ciphertext grows to size $N^{3/4}$. The only downside is that encryption times grow notably worse. When $a = 2/3$ or $a = 3/4$, this scheme has faster key generation times than the ABBE system from [GKSW10]. And for these values of a, decryption times are comparable, if not better, than those for [GKSW10], especially for the higher values of N.

At the end of the day, both this nonrisky broadcast and trace system and the ABBE system are viable options. This scheme allows for greater flexibility because the value of a can be changed, but the downside is that encryption times are much slower than those for the ABBE system. The advantage is that for certain values of a, we can achieve more efficient key generation and decryption times. In addition, we can reduce public key and private key sizes to less than $O(\lambda \cdot \sqrt{N})$ by setting $a > 1/2$.

4 Applications of Broadcast Encryption

Online Video Streaming The most commonly referenced use case for broadcast encryption is online video streaming services like Netflix, Hulu, Quibi, etc. Users with permission are given access to a myriad of different videos, and ideally bandwidth usage and client side decryption processing requirements need to be minimized so that users can watch the videos in real time and can watch those videos on any device, regardless of processing capability. The user numbers for these services are vast. During the second quarter of 2020 [Cor20], Netflix and Hulu had at least 190 million and 30 million users respectively. For these media streaming cases, we would recommend using the classic Boneh-Gentry-Waters [BGW05] scheme as it provides the best combination of short ciphertexts and fast decryption times even for large user sets. For $N = 1$ million users, the [BGW05, $\mathcal{S}$3.1] variant provides the best bandwidth at 96B per ciphertext while decryption takes 350ms and the [BGW05, $\mathcal{S}$3.2] variant provides the fastest decryption at 1.6ms with a 32KB ciphertext. Either of these are reasonable choices, although if the bandwidth isn't a problem, we'd recommend [BGW05, $\mathcal{S}$3.2] due to its smaller public key size (of 160KB, whereas [BGW05, $\mathcal{S}$3.1] requires 128MB for 1 million users). For very large user datasets (e.g., in the 190 million range), using [BGW05, $\mathcal{S}$3.2] instead of [BGW05, $\mathcal{S}$3.1] becomes even more important, as the former's public keys scale with $\sqrt{N}$ while the latter scale with N. Both [BGW05] schemes were proven secure in the static attacker model; if one wants the stronger adaptive attacker security, Waters [Wat09] offers this and small 861B ciphertexts, although the public key sizes grow to 32MB for $N = 1$ million. Both of the identity-based systems [GW09, GW19] require *hours* to decrypt a single ciphertext when N is 1 million, so they are not contenders.

The performance hit from [GKSW09] (the best performing traitor tracing scheme) vs. [BGW05] could be worth it for the chance to combat revenue reducers like piracy. For $N = 1$ million users, the decryption time of [GKSW09] doubles (over [BGW05, $\mathcal{S}$3.2]) to 3.2ms while the ciphertext size grows by a factor of 19 to 605KB – larger, but reasonable on fast networks. The public key size roughly triples to 477KB. Zhandry's risky traitor tracing scheme [Zha20, $\mathcal{S}$9.3] provides the best bandwidth for tracing schemes at only 384B, but the decryption time explodes to an infeasible 19 hours (for $N = 1$ million). Zhandry's nonrisky, post-user expansion compiler version of his scheme (see Section 3.8.10) has decryption times that are comparable to those for [GKSW09] and [GKSW10] for $N = 1$ million, but the encryption time balloons to over 1 minute and the ciphertext size jumps from roughly 600KB to almost 4MB (observe Table 16 when $a = 2/3$.) One potential benefit is that the public/private key sizes of Zhandry's Section 3.8.10 scheme are smaller, but that likely won't offset the additional encryption and bandwidth overhead.

Constraint Summary: The scheme needs to scale to 1 million users or more, with a small bandwidth overhead and fast (client side) decryption times. Encryption times are less of a concern, but traitor tracing may be needed.

Recommendation: Use [BGW05, $\mathcal{S}$3.2] (for fastest decryption and scalable public key size). See Section 3.3. If traitor tracing is required, use [GKSW09]. See Section 3.8.

Online Game Streaming Online game streaming is another form of media streaming that is becoming increasingly prevalent. Users receive high quality (resolution and frames per second) game data and give the server data like their keystrokes and mouse clicks in game. This system allows users to play games that have a performance requirement beyond what their client side device is capable of. In comparison to video streaming, online game streaming has a more stringent data speed requirement, as any wasted time could result in a subpar player experience. The most popular service, Nvidia GeForce Now, has around a million users [Nvi20], but it is a growing industry and the ceiling for game streaming services could be having user numbers on par with video streaming services. For this case, we would recommend [BGW05, $\mathcal{S}$3.2] or [Wat09]. Both schemes have ciphertexts under 1KB even for 1 million users, but the primary cost for both is a jump in public key size of 128MB and 32MB respectively. The scheme described in [Wat09] offers stronger provable security, while [BGW05, $\mathcal{S}$3.2] offers decryption times that are two orders of magnitude faster.

While the live traitor tracing functionality could be useful, the difference in performance could make a notable difference for users. Perhaps [GKSW09] could be used in situations where most of the game data is preloaded on the client side, and the live data is sent out live and unencrypted or from a faster performing scheme like [BGW05]. This could be a hybrid combination, allowing usage of the traitor tracing functionality, and ensuring fast enough performance. We note that the baseline ElGamal scheme takes almost 3 seconds to encrypt the payload for 1 million users, which likely rules this out for live gaming applications, highlighting the power of broadcast encryption for this setting.

Constraint Summary: The system needs to scale to 1 million users or more, with a combination of bandwidth overhead and (client side) decryption times that support live interactions. Overall, it needs to balance the benefits of traitor tracing with impact on user experience.

Recommendation: Use [BGW05, $\mathcal{S}$3.2] (for fastest decryption). If faster transmission is needed for the live gaming experience, use [BGW05, $\mathcal{S}$3.1] (shortest ciphertext, but largest PK) or [Wat09] (short ciphertext and more tolerable PK size) to cut the bandwidth overhead. See Tables 8 and 9. The overhead required for traitor tracing may frustrate the live gaming experience, but if it is needed, a hybrid approach using [GKSW09] for the most sensitive data may work. See Tables 15 and 14.

Live Sports Betting A novel use case for broadcast encryption arrives with the emergence of live sports betting. Due to the new developments in wireless data speeds with the emergence of 5G technologies, some major companies are developing capabilities for in-person spectators to make bets on their mobile phones throughout a game, utilizing continually updating betting lines given the events happening within the game. Broadcast encryption could be used to quickly send out information to users about how much current bets are worth to cash out and the current betting lines, all in realtime. In this use case, the total speed is the

most important factor (making the encryption time more relevant here), and the total number of users is within a pretty regular range ($N = 30{,}000$ to $70{,}000$), which is much smaller than the user amounts in some other use cases. Like with online game streaming, broadcast encryption offers real performance savings over individually encrypting with ElGamal; when $N = 100,000$ the encryption plus decryption time of ElGamal is roughly 10 times that of [GW09, $\mathcal{S}$3.1] or [BGW05, $\mathcal{S}$3.1]. For the $N \leq 100,000$ range, the public keys of [BGW05, $\mathcal{S}$3.1] are 13MB, while the public keys of [GW09, $\mathcal{S}$3.1] are a more tolerable 3MB. Systems [GW09, $\mathcal{S}$3.1] and [BGW05, $\mathcal{S}$3.1] tie for the shortest ciphertexts at 96B. The fastest encryption plus decryption time is [GW09] for this user level (and this holds over a range of sizes of allowed decrypter sets S from 10% to 50% of N), although the difference (a few milliseonds) isn't likely to be observable by a human.

Constraint Summary: We are looking for a sweet spot in the 10,000 to 100,000 user range, with a combination of bandwidth overhead and (server side) encryption and (client side) decryption times that support real-time interactions.

Recommendation: Use [GW09, $\mathcal{S}$3.1] (for best bandwidth, best sum of encryption and decryption time, and public key size tolerable for $N \leq 100,000$). See Section 3.4. The overhead required for traitor tracing may frustrate the live betting experience, but if it is needed, a hybrid approach using both [GKSW09] and [GW09] may work. See Section 3.8.

Distributor Limited Applications In the above applications, we assume that the distributor (e.g., Netflix, Disney) has large computing resources at its disposal. However, we also anticipate use cases where distributor performance becomes a bottle-neck (e.g., where a person is streaming video from their smartphone to a group). Possibly in the case of a direct peer-to-many-peers type of communication, the performance of the distributor system becomes relevant, thus making the times for the Setup, KeyGen and Encrypt functions more critical. In this situation, a simple recommendation is harder to make. [BGW05, $\mathcal{S}$3.2] for example, performs the best in the case of online video streaming, but if one person was streaming video from their smartphone directly to many peers, the encryption performance of [BGW05, $\mathcal{S}$3.2] is much worse than [BGW05, $\mathcal{S}$3.1] and [GW09, $\mathcal{S}$3.1]. Due to that constraint, if there are limited resources for the distributor, using [BGW05, $\mathcal{S}$3.2] isn't a good choice. If peers are less than 100K, in a situation with a distributor bottleneck we'd recommend either [BGW05, $\mathcal{S}$3.1] or [GW09, $\mathcal{S}$3.1]. The latter has much better performance in terms of encryption times, but the prior is orders of magnitude faster during KeyGen. In a situation where many keys are regularly generated, [BGW05, $\mathcal{S}$3.1] would be preferable, but in cases where keys are generated less often [GW09, $\mathcal{S}$3.1] will have the best performance, allowing the fastest encryption.

The same consideration can be made for the traitor tracing schemes. When the amount of users is around 1K, [GKSW10] slightly outperforms [GKSW09] in both setup times and encryption times, with roughly the same decryption times and notably worse KeyGen times. In a situation with distributor performance restraints, where many keys are regularly generated, [GKSW09] will perform better. In a situation where keys are generated less often and there are less than 1K users, [GKSW10] can perform better. However, both of these schemes are outperformed by the baseline (individual encryption to each peer) until about about the 10K user level.

Constraint Summary: For the 1K to 100K user range, we are looking for a scheme that optimizes the distributor functions without sacrificing much bandwidth or client performance.

Recommendation: Use [BGW05, $\mathcal{S}$3.1] (if need to generate keys often) or [GW09, $\mathcal{S}$3.1] (otherwise). See Tables 8 and 9. If traitor tracing is required for under 10K users, the baseline (individual encryption) will likely outperform any of the tracing broadcast schemes. If traitor tracing is required for over 10K users, use [GKSW09]. Compare Tables 8 and 9 to Tables 15 and 14.

5 Conclusion

Broadcast encryption allows sensitive data to be securely transmitted to a subset of users (who are all listening on a broadcast channel) more efficiently than individually encrypting to each user. This implementation study provides concrete implementation details and performance measurements for a host of pairing-based

broadcast encryption systems. It demonstrates the significant performance gains possible from broadcast encryption once the number of system users exceeds 10,000. It highlights the different broadcast systems that best fit the demanding settings of video streaming, online gaming, live sports betting and distributor limited applications.

6 Acknowledgments

The authors are grateful to Mark Zhandry for helpful interactions regarding his work [Zha20] and Brent Waters for helpful discussions regarding prior work in broadcast encryption.

Susan Hohenberger was supported by NSF CNS-1908181, the Office of Naval Research N00014-19-1-2294, and a Packard Foundation Subaward via UT Austin. Satyanarayana Vusirikala was supported by a UT Austin Provost Fellowship, NSF CNS-1908611, and the Packard Foundation.

References

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *CRYPTO*, pages 258–275. Springer, 2005.

[BS02] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *IACR Cryptol. ePrint Arch.*, 2002:80, 2002.

[BSW06] Dan Boneh, Amit Sahai, and Brent Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In *EUROCRYPT*, pages 573–592, 2006.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334, 2007.

[BW06] Dan Boneh and Brent Waters. A fully collusion resistant broadcast, trace, and revoke system. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS*, pages 211–220. ACM, 2006.

[BWZ14] Dan Boneh, Brent Waters, and Mark Zhandry. Low overhead broadcast encryption from multilinear maps. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 206–223. Springer, 2014.

[CFN94] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In *CRYPTO*, pages 257–270, 1994.

[Com19] The Walt Disney Company. Fiscal year 2019 annual financial report, 2019.

[Cor20] Netflix Corporation. Netflix q2 2020 shareholder letter, July 16, 2020.

[DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM J. Computing*, 30(2):391–437, 2000.

[Del07] Cécile Delerablée. Identity-based broadcast encryption with constant size ciphertexts and private keys. In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, 2007.

[DF02] Yevgeniy Dodis and Nelly Fazio. Public key broadcast encryption for stateless receivers. In Joan Feigenbaum, editor, *Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002, Washington, DC, USA, November 18, 2002, Revised Papers*, volume 2696 of *Lecture Notes in Computer Science*, pages 61–80. Springer, 2002.

[FN93] Amos Fiat and Moni Naor. Broadcast encryption. In *CRYPTO 1993*, 1993.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. How to enhance the security of public-key encryption at minimum cost. In *International Workshop on Public Key Cryptography*, pages 53–68. Springer, 1999.

[Gam85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory*, 31(4):469–472, 1985.

[GKRW18] Rishab Goyal, Venkata Koppula, Andrew Russell, and Brent Waters. Risky traitor tracing and new differential privacy negative results. In *CRYPTO 2018*, 2018.

[GKSW09] Sanjam Garg, Abishek Kumarasubramanian, Amit Sahai, and Brent Waters. Building efficient fully collusion-resilient traitor tracing and revocation schemes. *IACR Cryptol. ePrint Arch.*, 2009:532, 2009.

[GKSW10] Sanjam Garg, Abishek Kumarasubramanian, Amit Sahai, and Brent Waters. Building efficient fully collusion-resilient traitor tracing and revocation schemes. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 121–130. ACM, 2010.

[GKW17] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 612–621, 2017.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Jour. of Computer and System Science*, 28(2):270–299, 1984.

[GQWW19] Rishab Goyal, Willy Quach, Brent Waters, and Daniel Wichs. Broadcast and trace with n$\hat{}\epsilon$ ciphertext size from standard assumptions. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 826–855. Springer, 2019.

[GST04] Michael T. Goodrich, Jonathan Z. Sun, and Roberto Tamassia. Efficient tree-based revocation in groups of low-state devices. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 511–527. Springer, 2004.

[GW09] Craig Gentry and Brent Waters. Adaptive security in broadcast encryption systems (with short ciphertexts). In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 171–188. Springer, 2009.

[GW19] Aijun Ge and Puwen Wei. Identity-based broadcast encryption with efficient revocation. In Dongdai Lin and Kazue Sako, editors, *Public-Key Cryptography - PKC 2019 - 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part I*, volume 11442 of *Lecture Notes in Computer Science*, pages 405–435. Springer, 2019.

[Hor03] Jeremy Horwitz. A survey of broadcast encryption. Technical report, 2003.

[LLW+16] Weiran Liu, Jianwei Liu, Qianhong Wu, Bo Qin, David Naccache, and Houda Ferradi. Compact cca2-secure hierarchical identity-based broadcast encryption for fuzzy-entity data sharing. *IACR Cryptol. ePrint Arch.*, 2016:634, 2016.

[LSW10] Allison B. Lewko, Amit Sahai, and Brent Waters. Revocation systems with very small private keys. In *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA*, pages 273–285. IEEE Computer Society, 2010.

[NNL02] Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and tracing schemes for stateless receivers. *Electron. Colloquium Comput. Complex.*, 043, 2002.

[Nvi20] Nvidia. 2020 nvidia corporation annual review, 2020.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC*, pages 427–437, 1990.

[RS91] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, pages 433–444, 1991.

[RS16] Somindu C. Ramanna and Palash Sarkar. Efficient adaptively secure IBBE from the SXDH assumption. *IEEE Trans. Inf. Theory*, 62(10):5709–5726, 2016.

[SF07] Ryuichi Sakai and Jun Furukawa. Identity-based broadcast encryption. *IACR Cryptol. ePrint Arch.*, 2007.

[Wat09] Brent Waters. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In *CRYPTO*, pages 619–636. Springer, 2009.

[WB10] Jin Wang and Jingguo Bi. Lattice-based identity-based broadcast encryption scheme. *IACR Cryptol. ePrint Arch.*, 2010:288, 2010.

[WWW15] Fenghe Wang, Xu An Wang, and Chunxiao Wang. Lattice-based dynamical and anonymous broadcast encryption scheme. In Fatos Xhafa, Leonard Barolli, Fabrizio Messina, and Marek R. Ogiela, editors, *10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2015, Krakow, Poland, November 4-6, 2015*, pages 853–858. IEEE Computer Society, 2015.

[Zha20] Mark Zhandry. New techniques for traitor tracing: Size $n^{1/3}$ and more from pairings. In *CRYPTO*, volume 12170 of *Lecture Notes in Computer Science*, pages 652–682. Springer, 2020. See full version at https://eprint.iacr.org/2020/954.pdf.

Item	Scheme	Time when number of system users $N =$				
		100	1K	10K	100K	1M
Setup	baseline 3.2	4.46 ms	27.39 ms	252.65 ms	2.40 s	23.21 s
	[BGW05, $\mathcal{S}$3.1] 3.3.1	51.47 ms	456.97 ms	4.94 s	40.06 s	7.06 min
	[BGW05, $\mathcal{S}$3.2] 3.3.2	14.09 ms	15.75 ms	58.81 ms	158.08 ms	762.45 ms
	[GW09, $\mathcal{S}$3.1] 3.4	35.00 ms	69.40 ms	321.98 ms	2.80 s	29.44 s
	[Wat09] 3.5	32.39 ms	34.45 ms	297.71 ms	3.05 s	29.71 s
KeyGen	baseline 3.2	—	—	—	—	—
	[BGW05, $\mathcal{S}$3.1] 3.3.1	0.19 ms	0.18 ms	0.09 ms	0.11 ms	0.12 ms
	[BGW05, $\mathcal{S}$3.2] 3.3.2	0.11 ms	0.10 ms	0.14 ms	0.12 ms	0.16 ms
	[GW09, $\mathcal{S}$3.1] 3.4	11.07 ms	164.14 ms	954.00 ms	9.75 s	1.77 min
	[Wat09] 3.5	11.05 ms	104.49 ms	954.14 ms	9.78 s	1.60 min
Enc	baseline 3.2	0.39 ms	2.85 ms	25.52 ms	248.08 ms	2.81 s
	[BGW05, $\mathcal{S}$3.1] 3.3.1	1.26 ms	1.32 ms	2.40 ms	16.35 ms	664.76 ms
	[BGW05, $\mathcal{S}$3.2] 3.3.2	3.58 ms	8.61 ms	26.60 ms	86.36 ms	306.96 ms
	[GW09, $\mathcal{S}$3.1] 3.4	1.18 ms	1.37 ms	1.99 ms	12.63 ms	153.56 ms
	[Wat09] 3.5	2.03 ms	2.38 ms	3.44 ms	14.55 ms	131.80 ms
Dec	baseline 3.2	0.04 ms	0.03 ms	0.03 ms	0.05 ms	0.07 ms
	[BGW05, $\mathcal{S}$3.1] 3.3.1	2.60 ms	1.92 ms	2.78 ms	15.36 ms	349.52 ms
	[BGW05, $\mathcal{S}$3.2] 3.3.2	2.13 ms	1.49 ms	2.12 ms	2.46 ms	1.63 ms
	[GW09, $\mathcal{S}$3.1] 3.4	2.82 ms	2.82 ms	2.93 ms	16.56 ms	261.98 ms
	[Wat09] 3.5	6.37 ms	6.44 ms	7.49 ms	23.05 ms	157.66 ms

Table 8: Time evaluation for general public key broadcast encryption systems. The baseline scheme was implemented using NIST P-256 in OpenSSL, while the pairing-based schemes used curve BN254 in MCL. The KeyGen and Dec times represent the cost for a single user, while the Setup is the cost to initialize the entire system and Enc is the cost to encrypt to an arbitrary 10% of the system users. Let ms denote milliseconds, s denote seconds, and min denote minutes.

Item	Scheme	Space when number of system users $N =$				
		100	1K	10K	100K	1M
pk	baseline 3.2	3.23 KB	32.03 KB	320.03 KB	3.20 MB	32.00 MB
	[BGW05, $\mathcal{S}$3.1] 3.3.1	12.90 KB	128.10 KB	1.28 MB	12.80 MB	128.00 MB
	[BGW05, $\mathcal{S}$3.2] 3.3.2	1.66 KB	5.09 KB	16.06 KB	50.66 KB	160.06 KB
	[GW09, $\mathcal{S}$3.1] 3.4	3.68 KB	32.48 KB	320.48 KB	3.20 MB	32.00 MB
	[Wat09] 3.5	4.19 KB	32.99 KB	320.99 KB	3.20 MB	32.00 MB
sk	baseline 3.2	32.00 B	32.00 B	32.00 B	32.00 B	32.00 B
	[BGW05, $\mathcal{S}$3.1] 3.3.1	32.00 B	32.00 B	32.00 B	32.00 B	32.00 B
	[BGW05, $\mathcal{S}$3.2] 3.3.2	32.00 B	32.00 B	32.00 B	32.00 B	32.00 B
	[GW09, $\mathcal{S}$3.1] 3.4	3.26 KB	32.06 KB	320.06 KB	3.20 MB	32.00 MB
	[Wat09] 3.5	3.49 KB	32.29 KB	320.29 KB	3.20 MB	32.00 MB
ct	baseline 3.2	352.00 B	3.23 KB	32.03 KB	320.03 KB	3.20 MB
	[BGW05, $\mathcal{S}$3.1] 3.3.1	96.00 B	96.00 B	96.00 B	96.00 B	96.00 B
	[BGW05, $\mathcal{S}$3.2] 3.3.2	384.00 B	1.12 KB	3.26 KB	10.21 KB	32.06 KB
	[GW09, $\mathcal{S}$3.1] 3.4	96.00 B	96.00 B	96.00 B	96.00 B	96.00 B
	[Wat09] 3.5	861.00 B	861.00 B	861.00 B	861.00 B	861.00 B

Table 9: Space evaluation for general public key broadcast encryption systems. In the above, we set $|S|$, the size of the set of users a ciphertext is encrypted to, to be an arbitrary 10% of the total number of system users. The baseline scheme was implemented using NIST P-256 in OpenSSL, while the pairing-based schemes used curve BN254 in MCL. Let B denote bytes, KB denote kilobytes, and MB denote megabytes.

Scheme	Operation Count			
	Setup	KeyGen	Enc	Dec
baseline 3.2	${\lambda_0}^3 \cdot N$	—	${\lambda_0}^3 \cdot (\lvert S\rvert + 1) + \lambda_0 \cdot \lvert S\rvert$	$2 \cdot {\lambda_0}^3 + \lambda_0$
[BGW05, $\mathcal{S}$3.1] 3.3.1	${\lambda_1}^3 \cdot (2N - 1) + {\lambda_2}^3 \cdot N + e$	${\lambda_1}^3$	$\lambda_1 \cdot \lvert S\rvert + {\lambda_1}^3 + {\lambda_2}^3$	$\lambda_1 \cdot (\lvert S\rvert - 1) + 2 \cdot e$
[BGW05, $\mathcal{S}$3.2] 3.3.2	${\lambda_1}^3 \cdot (2B + A - 1) + {\lambda_2}^3 \cdot B + e$	${\lambda_1}^3$	${\lambda_1}^3 \cdot A + \lambda_1 \cdot \lvert S\rvert + {\lambda_2}^3$	$\lambda_1 \cdot \lvert S_a\rvert + 2 \cdot e$
[GW09, $\mathcal{S}$3.1] 3.4	$2 \cdot e + {\lambda_1}^3 + {\lambda_2}^3$	${\lambda_1}^3 \cdot N + {\lambda_2}^3$	$\lambda_1 \cdot \lvert S\rvert + {\lambda_1}^3 + {\lambda_2}^3$	$\lambda_1 \cdot (\lvert S\rvert - 1) + 2 \cdot e$
[Wat09] 3.5	$7 \cdot {\lambda_1}^3 + 6 \cdot {\lambda_2}^3 + 2 \cdot e + 2 \cdot \lambda_1$	${\lambda_1}^3 \cdot (N + 8) + 2 \cdot {\lambda_2}^3 + 5 \cdot \lambda_1$	$\lambda_1 \cdot (\lvert S\rvert + 2) + 6 \cdot {\lambda_1}^3 + 6 \cdot {\lambda_2}^3$	$\lambda_1 \cdot (\lvert S\rvert - 1) + 9 \cdot e$

Table 10: Operation counts, where N is the total number of users in the broadcast system.

Scheme	Theoretical Runtime			
	Setup	KeyGen	Enc	Dec
baseline 3.2	$O(\lambda^3 \cdot N)$	—	$O(\lambda^3 \cdot \|S\| + \lambda \cdot \|S\|)$	$O(\lambda)$
[BGW05, $\mathcal{S}$3.1] 3.3.1	$O(\lambda^3 \cdot N)$	$O(\lambda^3)$	$O(\lambda \cdot \|S\|)$	$O(\lambda \cdot \|S\|)$
[BGW05, $\mathcal{S}$3.2] 3.3.2	$O(\lambda^3 \cdot \sqrt{N})$	$O(\lambda^3)$	$O(N + \lambda^3 \cdot \sqrt{N} + \lambda \cdot \|S\|)$	$O(\lambda \cdot \|S_a\|)$
[GW09, $\mathcal{S}$3.1] 3.4	$O(\delta_1 \cdot N)$	$O(\lambda^3 \cdot N)$	$O(\lambda \cdot \|S\|)$	$O(\lambda \cdot \|S\|)$
[Wat09] 3.5	$O(\delta_1 \cdot N)$	$O(\lambda^3 \cdot N)$	$O(\lambda \cdot \|S\|)$	$O(\lambda \cdot \|S\|)$

Table 11: Theoretical runtimes, where N is the total number of users in the broadcast system.

Item	Scheme	Time when number of system users $N =$				
		100	1K	10K	100K	1M
Setup	[GW09, $\mathcal{S}$4.1]	13.41 ms	68.30 ms	554.30 ms	6.64 s	59.99 s
	[GW09, $\mathcal{S}$4.3.1]	4.84 ms	48.19 ms	329.00 ms	3.19 s	33.56 s
	[GW19, $\mathcal{S}$3.1]	93.34 ms	415.42 ms	3.85 s	40.64 s	6.84 min
KeyGen	[GW09, $\mathcal{S}$4.1]	0.47 ms	0.31 ms	0.26 ms	0.35 ms	0.31 ms
	[GW09, $\mathcal{S}$4.3.1]	0.47 ms	0.27 ms	0.51 ms	1.31 ms	2.06 ms
	[GW19, $\mathcal{S}$3.1]	101.77 ms	563.62 ms	6.28 s	1.08 minutes	9.36 min
Enc	[GW09, $\mathcal{S}$4.1]	8.46 ms	48.78 ms	3.40 s	2.66 min	4.87 hrs
	[GW09, $\mathcal{S}$4.3.1]	4.61 ms	45.35 ms	2.90 s	2.64 min	4.73 hrs
	[GW19, $\mathcal{S}$3.1]	5.54 ms	29.00 ms	3.13 s	2.85 min	4.57 hrs
Dec	[GW09, $\mathcal{S}$4.1]	9.84 ms	46.14 ms	2.32 s	2.70 min	4.91 hrs
	[GW09, $\mathcal{S}$4.3.1]	6.60 ms	28.95 ms	1.99 s	2.62 min	4.73 hrs
	[GW19, $\mathcal{S}$3.1]	8.21 ms	32.45 ms	1.52 s	2.81 min	4.56 hrs

Table 12: Time Evaluation for IBBE Systems. All the IBBE schemes were tested over the curve BN254 in MCL. The KeyGen and Dec times represent the cost for a single user, while the Setup is the cost to initialize the entire system and Enc is the cost to encrypt to an arbitrary 10% of the system users. Let ms denote milliseconds, s denote seconds, min denote minutes, and hrs denote hours.

Item	Scheme	Space when number of system users $N =$				
		100	1K	10K	100K	1M
pk	[GW09, $\mathcal{S}$4.1]	1.92 KB	19.2 KB	192 KB	1.92 MB	19.20 MB
	[GW09, $\mathcal{S}$4.3.1]	1.06 KB	9.70 KB	96.10 KB	960.10 KB	9.60 MB
	[GW19, $\mathcal{S}$3.1]	7.36 KB	64.96 KB	640.96 KB	6.40 MB	64.00 MB
sk	[GW09, $\mathcal{S}$4.1]	95.75 B	95.75 B	95.75 B	95.75 B	95.75 B
	[GW09, $\mathcal{S}$4.3.1]	95.75 B	95.75 B	95.75 B	95.75 B	95.75 B
	[GW19, $\mathcal{S}$3.1]	9.67 KB	95.85 KB	957.60 KB	9.58 MB	95.75 MB
ct	[GW09, $\mathcal{S}$4.1]	762.75 B	3.62 KB	32.20 KB	317.95 KB	3.18 MB
	[GW09, $\mathcal{S}$4.3.1]	477.00 B	477.00 B	477.00 B	477.00 B	477.00 B
	[GW19, $\mathcal{S}$3.1]	192.00 B	192.00 B	192.00 B	192.00 B	192.00 B

Table 13: Space Evaluation for IBBE Systems. In the above, we set $|S|$, the size of the set of users a ciphertext is encrypted to, to be an arbitrary 10% of the total number of system users. All the sizes are measured over the curve BN254 in MCL. Let B denote bytes, KB denote kilobytes, and MB denote megabytes.

Item	Scheme	Time when number of system users $N =$				
		100	1K	10K	100K	1M
Setup	[GKSW09]	13.45 ms	25.67 ms	66.95 ms	476.00 ms	692.90 ms
	[GKSW10]	12.24 ms	23.92 ms	74.52 ms	223.11 ms	586.45 ms
	[Zha20, $\mathcal{S}$9.3]	118.97 ms	870.93 ms	8.60 s	1.47 min	15.79 min
KeyGen	[GKSW09]	0.74 ms	0.54 ms	0.50 ms	0.74 ms	0.50 ms
	[GKSW10]	1.25 ms	3.15 ms	9.61 ms	26.63 ms	97.63 ms
	[Zha20, $\mathcal{S}$9.3]	0.95 ms	1.55 ms	0.80 ms	2.13 ms	9.59 ms
Enc	[GKSW09]	26.84 ms	78.34 ms	190.12 ms	771.29 ms	2.06 s
	[GKSW10]	19.48 ms	68.62 ms	227.05 ms	733.95 ms	4.77 s
	[Zha20, $\mathcal{S}$9.3]	7.53 ms	95.77 ms	3.34 s	2.79 min	19.34 hrs
Dec	[GKSW09]	4.15 ms	3.48 ms	3.20 ms	3.97 ms	3.21 ms
	[GKSW10]	3.41 ms	3.51 ms	4.16 ms	4.52 ms	9.58 ms
	[Zha20, $\mathcal{S}$9.3]	11.56 ms	111.84 ms	3.31 s	2.69 min	19.19 hrs

Table 14: Time Evaluation for Tracing Systems. All the tracing schemes were tested over the curve BN254 in MCL. The KeyGen and Dec times represent the cost for a single user, while the Setup is the cost to initialize the entire system. For [GKSW10] and [Zha20, $\mathcal{S}$9.3], Enc is the cost to encrypt to an arbitrary 10% of the total number of system users. For the PLBE system in [GKSW09], Enc is the cost to encrypt to all of the system users. Let ms denote milliseconds, s denote seconds, min denote minutes, and hrs denote hours.

Item	Scheme	Space when number of system users $N =$				
		100	1K	10K	100K	1M
pk	[GKSW09]	4.87 KB	15.36 KB	47.80 KB	151.31 KB	477.10 KB
	[GKSW10]	5.19 KB	16.39 KB	51.00 KB	161.45 KB	509.10 KB
	[Zha20, $\mathcal{S}$9.3]	13.57 KB	128.77 KB	1.28 MB	12.80 MB	128.00 MB
sk	[GKSW09]	64.00 B	64.00 B	64.00 B	64.00 B	64.00 B
	[GKSW10]	384.00 B	1.09 KB	3.26 KB	10.21 KB	32.06 KB
	[Zha20, $\mathcal{S}$9.3]	256.00 B	256.00 B	256.00 B	256.00 B	256.00 B
ct	[GKSW09]	6.05 KB	19.36 KB	60.50 KB	191.79 KB	605.00 KB
	[GKSW10]	6.69 KB	21.41 KB	66.90 KB	212.07 KB	669.00 KB
	[Zha20, $\mathcal{S}$9.3]	384.00 B	384.00 B	384.00 B	384.00 B	384.00 B

Table 15: Space Evaluation for Tracing Systems. For [GKSW10] and [Zha20, $\mathcal{S}$9.3], we set $|S|$, the size of the set of privileged users, to be an arbitrary 10% of the total number of system users. The PLBE system in [GKSW09] relies on private tracing, and, therefore, does not contain a privileged user subset. All the sizes are measured over the curve BN254 in MCL. Let B denote bytes, KB denote kilobytes, and MB denote megabytes.

Item	$a =$	Time when number of system users $N =$				
		100	1K	10K	100K	1M
Setup	1/3	38.90 ms	120.73 ms	411.34 ms	2.70 s	9.60 s
	1/2	13.79 ms	38.13 ms	101.73 ms	346.40 ms	1.99 s
	2/3	11.90 ms	15.31 ms	24.77 ms	63.19 ms	180.92 ms
	3/4	14.29 ms	12.55 ms	18.80 ms	32.41 ms	147.08 ms
	1	7.02 ms	7.94 ms	13.05 ms	17.82 ms	68.51 ms
KeyGen	1/3	24.90 ms	91.48 ms	748.18 ms	2.34 s	10.76 s
	1/2	14.92 ms	30.38 ms	81.37 ms	273.22 ms	730.97 ms
	2/3	6.05 ms	8.08 ms	21.25 ms	33.68 ms	79.52 ms
	3/4	5.02 ms	4.78 ms	7.42 ms	14.06 ms	23.54 ms
	1	0.77 ms	1.47 ms	0.80 ms	0.81 ms	0.91 ms
Enc	1/3	10.42 ms	61.20 ms	479.07 ms	10.58 s	3.26 min
	1/2	18.47 ms	109.24 ms	599.37 ms	5.46 s	1.03 min
	2/3	41.20 ms	187.05 ms	1.40 s	7.53 s	1.04 min
	3/4	96.50 ms	327.75 ms	1.95 s	13.41 s	1.58 min
	1	179.51 ms	1.98 s	16.69 s	2.78 min	—
Dec	1/3	9.88 ms	15.35 ms	60.30 ms	236.07 ms	1.85 s
	1/2	7.72 ms	9.46 ms	12.75 ms	20.37 ms	48.99 ms
	2/3	12.10 ms	7.62 ms	10.23 ms	9.60 ms	10.52 ms
	3/4	13.16 ms	9.02 ms	7.80 ms	8.98 ms	8.03 ms
	1	6.77 ms	12.85 ms	6.44 ms	6.43 ms	—

Table 16: Time Evaluation for the non-risky broadcast multi-scheme for different values of a. The KeyGen and Dec are the time costs for a single user, while the Setup is the cost to initialize the entire system and Enc is the cost to encrypt to an arbitrary 10% of the system users.

Item	$a =$	Space when number of system users $N =$				
		100	1K	10K	100K	1M
pk	1/3	3.97 KB	13.57 KB	61.82 KB	279.04 KB	1.28 MB
	1/2	2.05 KB	4.99 KB	13.57 KB	41.34 KB	128.77 KB
	2/3	1.41 KB	2.05 KB	3.58 KB	6.78 KB	13.57 KB
	3/4	1.28 KB	1.53 KB	2.05 KB	3.07 KB	4.86 KB
	1	893.00 B	893.00 B	893.00 B	893.00 B	893.00 B
sk	1/3	6.40 KB	25.60 KB	122.11 KB	556.54 KB	2.56 MB
	1/2	2.56 KB	8.45 KB	25.60 KB	81.15 KB	256.00 KB
	2/3	1.28 KB	2.56 KB	5.63 KB	12.03 KB	25.60 KB
	3/4	1.02 KB	1.54 KB	2.56 KB	4.61 KB	8.19 KB
	1	256.00 B	256.00 B	256.00 B	256.00 B	256.00 B
ct	1/3	1.54 KB	3.84 KB	8.06 KB	17.66 KB	38.40 KB
	1/2	3.84 KB	11.90 KB	38.40 KB	121.34 KB	384.00 KB
	2/3	8.06 KB	38.40 KB	178.18 KB	827.14 KB	3.84 MB
	3/4	11.90 KB	67.97 KB	384.00 KB	2.16 MB	12.14 MB
	1	38.40 KB	384.00 KB	3.84 MB	38.40 MB	384.00 MB

Table 17: Space Evaluation for the non-risky broadcast multi-scheme for different values of a, where the public key, private key, and ciphertext have size N^{1-a}, N^{1-a}, and N^a, respectively. In the above, we set $|S|$, the size of the set of users a ciphertext is encrypted to, to be an arbitrary 10% of the total number of system users. All the sizes are measured over the curve BN254 in MCL. Let B denote bytes, KB denote kilobytes, and MB denote megabytes.