A Performance Evaluation of Pairing-Based Broadcast Encryption Systems

Arush Chhatrapati[1], Susan Hohenberger[2(✉)], James Trombo[3], and Satyanarayana Vusirikala[4]

[1] Henry M. Gunn High School, Palo Alto, CA, USA
[2] Johns Hopkins University, Baltimore, MD, USA
susan@cs.jhu.edu
[3] George Mason High School, Falls Church, VA, USA
[4] University of Texas at Austin, Austin, TX, USA
satya@cs.utexas.edu

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 is to leverage the broadcasting structure to achieve better efficiency than individually encrypting to each user; in particular, reducing the 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 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 distributor-limited applications. This work can inform both practitioners and future cryptographic designers in this area.

1 Introduction

In a broadcast encryption system [9], a sender can encrypt a message for any subset of users who are listening on a broadcast channel. We focus on public-key

S. Hohenberger—Supported by NSF CNS-1908181, the Office of Naval Research N00014-19-1-2294, and a Packard Foundation Subaward via UT Austin.

S. Vusirikala—Supported by a UT Austin Provost Fellowship, NSF CNS-1908611, and the Packard Foundation.

© Springer Nature Switzerland AG 2022

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 ciphertext size, public and private key size, and the setup, key generation, encryption and decryption times. We focus on pairing-based broadcast systems, since this is the most promising algebraic setting for reducing ciphertext size and obtaining fast runtimes (see [4] for more on pairings). We also compare the broadcast schemes to an optimized "baseline" scheme[1] derived from ElGamal encryption [11] with shared parameters (see Sect. 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 eleven public-key, pairing-based broadcast encryption systems, which are detailed in Table 2 of Sect. 3 and which we thought were likely to perform the best. 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 herein, with details in the full version [4].
- We implemented the eleven broadcast systems using the MCL pairings library (currently employed by some cutting-edge cryptocurrency companies) and the baseline ElGamal system using OpenSSL. These implementations will be made publicly available. We ran hundreds of tests on these systems for various

[1] Because the Sect. 2 baseline scheme will not require the pairing operation, it is implemented using an elliptic curve group, whose elements are even smaller; thus requiring real performance gains from the broadcast systems to overtake it.

parameter choices, reporting on those results in Sect. 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 in a prior publication; we communicated it to the author(s) and they updated their scheme accordingly. (Details are removed for submission anonymity, but we will be explicit in the final version.) Thus, this implementation effort has also been useful as an additional verification process for prior work.

– In Tables 7 and 8, 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 [20], Disney [5] and Netflix [6] to understand the performance demands of some interesting applications for broadcast encryption. We summarize our findings in Sect. 4. We start with the classic application of video streaming and then explore the emerging applications of online gaming, live sports betting and more. In a nutshell, if traitor tracing is not required, we found that the classic Boneh-Gentry-Waters system [2, $\mathcal{S}$3.2] provides the best tradeoffs for video streaming and is strong for online gaming too, with [23] also strong for gaming. Zhandry's generalized nonrisky system (see [4] for details) 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 [15] the preferred choice. We found the private tracing system of Gentry, Kumarasubramanian, Sahai and Waters [12] 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 [2, $\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.

The schemes we implemented (as taken from their respective publications), including the ElGamal baseline, achieve security against chosen plaintext attacks [16] (CPA), while NIST recommends that deployed systems achieve a stronger notion of security against chosen ciphertext attacks [8,19,21] (CCA). While efficient general transformations from CPA to CCA exist for public key encryption [10], it is not clear if these can be applied to broadcast encryption systems without compromising some of their functionality (e.g., traitor tracing). This is an exciting area for future research.

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.

2 An ElGamal Baseline and Other Related Works

We construct a baseline system, which encrypts the same message individually to each privileged user, so that we can contrast its performance with the broadcast systems. This CPA-secure system uses ElGamal encryption [11] with shared parameters. Private key sizes are constant, but the public key size grows linearly with the number of system users.

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 tested 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.

Table 1. Runtimes for the ElGamal baseline over different curves when $N = 100$K and $|S| = 10$K. Let s denote seconds and ms denote milliseconds.

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

Additional related works are discussed in the full version [4].

3 Broadcast Encryption Implementations and Analysis

We refer to the following for the definitions of broadcast encryption [9], and the identity-based [7,22], trace-and-revoke [18], augmented [1], traitor-tracing [3,17], and private-linear [3] variants.

We provide a reference implementation[2] and comparison of eleven broadcast encryption systems. All schemes are implemented in the asymmetric pairing setting using the MCL pairings library[3] with a Barreto-Naehrig BN254 curve. This curve is conjectured to have approximately 100-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. We also compare all of the systems to an ElGamal baseline system in Sect. 2, which was implemented using prime-order elliptic curve groups in OpenSSL since it does not require pairings. The baseline system was implemented using C++ but all the others are in Java. We chose Java for the pairing-based broadcast schemes because the MCL Java library possessed a remarkably simple, flexible software interface which allowed us to easily implement and compare these systems to each other.

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 4 GB RAM.

We also compare the sizes of the public key, private key, and ciphertext for each of the systems. Table 2 shows how the sizes scale asymptotically. It also provides an overview of the systems that we implement.

3.1 Boneh-Gentry-Waters Scheme Using Asymmetric Pairings

The Boneh-Gentry-Waters-Scheme, [2], 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", [2, $\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 [2, $\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 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 in Sect. 3.4.

In Table 3, we present the encryption times for the BGW special case construction for varying subset sizes. We define the subset size to be some percent

[2] https://github.com/ArushC/broadcast.
[3] https://github.com/herumi/mcl.

Table 2. A summary of pairing-based broadcast encryption systems. 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 [24, $\mathcal{S}$9.3], $a \in [0, 1]$.

Scheme	Type	Ciphertext Size	Private Key Size	Public Key Size	Security
ElGamal baseline	Public Key	$O(\lambda \cdot \|S\|)$	$O(\lambda)$	$O(\lambda \cdot N)$	CPA-secure
[2, $\mathcal{S}$3.1]	Broadcast	$O(\lambda)$	$O(\lambda)$	$O(\lambda \cdot N)$	static
[2, $\mathcal{S}$3.2]	Broadcast	$O(\lambda \cdot \sqrt{N})$	$O(\lambda)$	$O(\lambda \cdot \sqrt{N})$	static
[23]	Broadcast	$O(\lambda)$	$O(\lambda \cdot N)$	$O(\lambda \cdot N)$	adaptive
[15, $\mathcal{S}$3.1]	Broadcast	$O(\lambda)$	$O(\lambda \cdot N)$	$O(\lambda \cdot N)$	semi-static
[15, $\mathcal{S}$4.1]	IBBE	$O(\lambda \cdot \ell)$	$O(\lambda)$	$O(\lambda \cdot \ell)$	adaptive
[15, $\mathcal{S}$4.3.1]	IBBE	$O(\lambda)$	$O(\lambda)$	$O(\lambda \cdot \ell)$	semi-static
[14, $\mathcal{S}$3.1]	IBBE	$O(\lambda)$	$O(\lambda \cdot N)$	$O(\lambda \cdot N)$	adaptive
[24, $\mathcal{S}$9.3]	Risky Trace	$O(\lambda \cdot N)$	$O(\lambda)$	$O(\lambda)$	adaptive
[13, $\mathcal{S}$5.2]	Trace	$O(\lambda \cdot \sqrt{N})$	$O(\lambda \cdot \sqrt{N})$	$O(\lambda \cdot \sqrt{N})$	adaptive and public tracing
[24, $\mathcal{S}$9.3]	Trace	$O(\lambda \cdot N^{1-a})$	$O(\lambda \cdot N^{1-a})$	$O(\lambda \cdot N^{a})$	adaptive
[12]	PLBE	$O(\lambda \cdot \sqrt{N})$	$O(\lambda)$	$O(\lambda \cdot \sqrt{N})$	private tracing

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.

We implement the general construction by setting $B = \lfloor \sqrt{n} \rfloor$ as the authors of [2] 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 instead of generating the private keys for all N users in the Setup phase.

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. To understand the context for the discussion that follows, we ask readers to refer to the bottom of page 7 of [2].

In the encryption algorithm, runtimes are determine 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 latter set, 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 $\ell = 1, 2, \ldots, A$. Therefore, the

Table 3. Encryption times for the Boneh-Gentry-Waters Special Case scheme. Let s denote seconds and ms denote milliseconds.

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

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, 1 K, 10 K), 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.

Table 4. Encryption times for the Boneh-Gentry-Waters general scheme. Let ms denote milliseconds.

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

3.2 Gentry-Waters: A Semi-static Variant of the BGW System

In [15], Gentry and Waters introduce the notion of semi-static security, which is between static security and adaptive security. They construct a semi-statically secure variant of [2]. 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. We implemented their semi-static scheme in the asymmetric (Type-III) pairing setting and optimized it for efficiency, with details in the full version. In Table 5, we give the encryption times. 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 $\mathbb{G}_1$.

Table 5. Encryption times for the Gentry-Waters semi-static variant of the Boneh-Gentry-Waters scheme. Let ms denote milliseconds.

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

3.3 Waters Dual System Broadcast Encryption System

We implement a broadcast encryption system that is secure against adaptive adversaries, described in [23]. 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 Sects. 3.1 and 3.2, 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. In the full version, we describe how we implemented it in the Type-III pairing setting, strategically choosing which group elements to place in $\mathbb{G}_1$ and $\mathbb{G}_2$ to maximize efficiency.

In Table 6, we show the encryption times from our implementation, which are dominated by the computation of $E_1 = (\prod_{i \in S} u_i)^t$, which requires $O(|S|)$ group multiplications over $\mathbb{G}_1$.

Table 6. Encryption times for the Waters Dual Broadcast System. Let ms denote milliseconds.

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

3.4 Comparison of General Broadcast Encryption Systems

We now compare the broadcast encryption systems that we describe in Sects. 3.1, 3.2, and 3.3 to each other, and to the baseline scheme which we describe in Sect. 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 asymptotic 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 our implementation 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 8. 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 2.

Setup Times. We start by analyzing the setup times presented in Table 7. 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, [2, $\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 on the order of $O(N)$ operations over $\mathbb{G}_1$ to calculate a linear-sized public key. From our implementation, individual group operations over the elliptic curve group $\mathbb{G}$ (NIST P-256) used for the baseline were found to be faster than operations over $\mathbb{G}_1$ used in the pairings-based schemes. This explains why the baseline setup times are faster than those for all of the pairings-based schemes *except* [2, $\mathcal{S}$3.2].

Setup times for [2, $\mathcal{S}$3.2] appear to be faster than those for the baseline when $N >= 1K$, but not when $N = 100$. This is because when $N = 100$, the difference in the total number of exponentiations computed during setup for the baseline and [2, $\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, [2, $\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.

Encryption Times. 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 at the baseline construction from Sect. 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 [2, $\mathcal{S}$3.2], the total number of exponentiations computed during encryption is constant. In [2, $\mathcal{S}$3.1] and [15, $\mathcal{S}$3.1], we only need one group exponentiation each time to compute $C_0 = (g_2)^t$. In [23], 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 [2, $\mathcal{S}$3.2]. Hence, all of the pairing-based schemes except [2, $\mathcal{S}$3.2] have faster encryption times when $N >= 1K$.
- If we compare the baseline to [2, $\mathcal{S}$3.2], we notice that [2, $\mathcal{S}$3.2] is only more efficient than the baseline when $N \geq 100K$. We recall that encryption in [2, $\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 [2, $\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 [2, $\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 [2, $\mathcal{S}$3.2].

– When we compare the pairing-based schemes to each other, we see that [15, $\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 [2, $\mathcal{S}$3.1]. As N grows larger, the encryption times for [23] grows closer to those for [15, $\mathcal{S}$3.1] and [2, $\mathcal{S}$3.1]. We recall that [15, $\mathcal{S}$3.1] is actually a semi-static variant of [2, $\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 that its static counterpart. So far, we see that the [15, $\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, [23] 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.

Key Generation Times. The baseline scheme, [2, $\mathcal{S}$3.1], and [2, $\mathcal{S}$3.1] 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 [15, $\mathcal{S}$3.1] and [23] 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 [15, $\mathcal{S}$3.1] and [23], it takes more than 1.5 min 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 require $O(N)$ operations over $\mathbb{G}_1$ to generate a single user's linear-sized private key. In [15, $\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 [23] 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. The keys for all the privileged users in S would take up $O(\lambda \cdot N \cdot |S|)$ space.

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 [2, $\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 [2, $\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

Table 7. 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	Time when number of system users $N =$				
		100	1K	10K	100K	1M
Setup	baseline 2	4.46 ms	27.39 ms	252.65 ms	2.40 s	23.21 s
	[2, $\mathcal{S}$3.1] 3.1	51.47 ms	456.97 ms	4.94 s	40.06 s	7.06 min
	[2, $\mathcal{S}$3.2] 3.1	14.09 ms	15.75 ms	58.81 ms	158.08 ms	762.45 ms
	[15, $\mathcal{S}$3.1] 3.2	35.00 ms	69.40 ms	321.98 ms	2.80 s	29.44 s
	[23] 3.3	32.39 ms	34.45 ms	297.71 ms	3.05 s	29.71 s
KeyGen	baseline 2	—	—	—	—	—
	[2, $\mathcal{S}$3.1] 3.1	0.19 ms	0.18 ms	0.09 ms	0.11 ms	0.12 ms
	[2, $\mathcal{S}$3.2] 3.1	0.11 ms	0.10 ms	0.14 ms	0.12 ms	0.16 ms
	[15, $\mathcal{S}$3.1] 3.2	11.07 ms	164.14 ms	954.00 ms	9.75 s	1.77 min
	[23] 3.3	11.05 ms	104.49 ms	954.14 ms	9.78 s	1.60 min
Enc	baseline 2	0.39 ms	2.85 ms	25.52 ms	248.08 ms	2.81 s
	[2, $\mathcal{S}$3.1] 3.1	1.26 ms	1.32 ms	2.40 ms	16.35 ms	664.76 ms
	[2, $\mathcal{S}$3.2] 3.1	3.58 ms	8.61 ms	26.60 ms	86.36 ms	306.96 ms
	[15, $\mathcal{S}$3.1] 3.2	1.18 ms	1.37 ms	1.99 ms	12.63 ms	153.56 ms
	[23] 3.3	2.03 ms	2.38 ms	3.44 ms	14.55 ms	131.80 ms
Dec	baseline 2	0.04 ms	0.03 ms	0.03 ms	0.05 ms	0.07 ms
	[2, $\mathcal{S}$3.1] 3.1	2.60 ms	1.92 ms	2.78 ms	15.36 ms	349.52 ms
	[2, $\mathcal{S}$3.2] 3.1	2.13 ms	1.49 ms	2.12 ms	2.46 ms	1.63 ms
	[15, $\mathcal{S}$3.1] 3.2	2.82 ms	2.82 ms	2.93 ms	16.56 ms	261.98 ms
	[23] 3.3	6.37 ms	6.44 ms	7.49 ms	23.05 ms	157.66 ms

decryption times. All of the pairing-based schemes except [23] required only two pairings to be computed during decryption. [23] required nine pairings. This large number of pairings explains why the decryption times are consistently the slowest for this system for $N <= 100K$. We again see that the decryption times for [2, $\mathcal{S}$3.1] are similar to those of its semi-static counterpart, [15, $\mathcal{S}$3.1]. And this makes sense. For both, decryption times are dominated by a step that requires $|S| - 1$ group multiplications over $\mathbb{G}_1$.

Overall Runtime Comparison. If we consider key generation a step in the decryption process, then [15, $\mathcal{S}$3.1] and [23] by far have the slowest runtimes. But recall that these are the only two systems that are secure against non-static

Table 8. 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.

Item	Scheme	Space when number of system users $N =$				
		100	1K	10K	100K	1M
pk	baseline 2	3.23 KB	32.03 KB	320.03 KB	3.20 MB	32.00 MB
	[2, $\mathcal{S}$3.1] 3.1	12.90 KB	128.10 KB	1.28 MB	12.80 MB	128.00 MB
	[2, $\mathcal{S}$3.2] 3.1	1.66 KB	5.09 KB	16.06 KB	50.66 KB	160.06 KB
	[15, $\mathcal{S}$3.1] 3.2	3.68 KB	32.48 KB	320.48 KB	3.20 MB	32.00 MB
	[23] 3.3	4.19 KB	32.99 KB	320.99 KB	3.20 MB	32.00 MB
sk	baseline 2	32.00 B	32.00 B	32.00 B	32.00 B	32.00 B
	[2, $\mathcal{S}$3.1] 3.1	32.00 B	32.00 B	32.00 B	32.00 B	32.00 B
	[2, $\mathcal{S}$3.2] 3.1	32.00 B	32.00 B	32.00 B	32.00 B	32.00 B
	[15, $\mathcal{S}$3.1] 3.2	3.26 KB	32.06 KB	320.06 KB	3.20 MB	32.00 MB
	[23] 3.3	3.49 KB	32.29 KB	320.29 KB	3.20 MB	32.00 MB
ct	baseline 2	352.00 B	3.23 KB	32.03 KB	320.03 KB	3.20 MB
	[2, $\mathcal{S}$3.1] 3.1	96.00 B	96.00 B	96.00 B	96.00 B	96.00 B
	[2, $\mathcal{S}$3.2] 3.1	384.00 B	1.12 KB	3.26 KB	10.21 KB	32.06 KB
	[15, $\mathcal{S}$3.1] 3.2	96.00 B	96.00 B	96.00 B	96.00 B	96.00 B
	[23] 3.3	861.00 B	861.00 B	861.00 B	861.00 B	861.00 B

adversaries. It is a tradeoff: in order to achieve the higher level of security, the decryption will be slower.

The encryption times for [15, $\mathcal{S}$3.1] and [23] 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.

Looking at Table 8, 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 [2, $\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 [15, $\mathcal{S}$3.1] and [23], they are still very fast in comparison to [2, $\mathcal{S}$3.1]. Additionally, the public key and private keys in [15, $\mathcal{S}$3.1] and [23] are all of size $O(\lambda \cdot N)$. This is very large. But we recall that while [2, $\mathcal{S}$3.1] and [2, $\mathcal{S}$3.2] are secure only against static adversaries, [15, $\mathcal{S}$3.1] is secure against semi-static adversaries and [23] 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 [15]. In general, the decryption times for [15, $\mathcal{S}$3.1] are much faster because they only require two applications of the pairing algorithm, while [23] requires nine pairings in decryption. Nevertheless, as the total number of system users N grows larger, the decryption times for [23] approach the times for [15, $\mathcal{S}$3.1]. So if we have a small total number of users in our system, we recommend [15, $\mathcal{S}$3.1]. But if the value of N is very large, then [23] will perform equally well during decryption. And because the adaptive security provided by [23] is stronger than the semi-static security provided by [15, $\mathcal{S}$3.1], we especially recommend [23] 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* [15, $\mathcal{S}$3.2]. Then, depending on the desired level of security, we would choose either the statically secure [2, $\mathcal{S}$3.1], the semi-statically secure [15, $\mathcal{S}$3.1], or the adaptively secure [23] broadcast system.

Further Theoretical Analysis. For further theoretical analysis, we denote λ_1 and λ_2 as a single group multiplication operation over $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. Exponentiations are denoted by ${\lambda_1}^3$ and ${\lambda_2}^3$. We let e denote a single pairing operation. For the baseline scheme, we simply use λ_0 and ${\lambda_0}^3$ to represent a single multiplication and exponentiation over the elliptic curve group $\mathbb{G}$, respectively. Here, 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 $O(X + \lambda \cdot Y)$, 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 9, 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 [23] broadcast system. Additionally, runtimes for the setup phase for [23] and [15, $\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 9. When we use big-O notation to describe the time that the setup phase took for these two schemes (see Table 10), 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 9. The big-O notation is best to refer to if we want to know which operation(s) are dominating runtimes, but for total runtime details Table 7 is better.

Table 9. Operation counts, where N is the total number of users in the system.

Scheme	Operation Count			
	Setup	KeyGen	Enc	Dec
baseline 2	${\lambda_0}^3 \cdot N$	—	${\lambda_0}^3 \cdot (\lvert S\rvert + 1) + \lambda_0 \cdot \lvert S\rvert$	$2 \cdot {\lambda_0}^3 + \lambda_0$
[2, $\mathcal{S}$3.1] 3.1	${\lambda_1}^3 \cdot (2N - 1) + {\lambda_2}^3 \cdot N + e$	${\lambda_1}^3$	$\lambda_1 \cdot \lvert S\rvert + {\lambda_1}^3 + {\lambda_2}^3$	$\lambda_1 \cdot (\lvert S\rvert - 1) + 2 \cdot e$
[2, $\mathcal{S}$3.2] 3.1	${\lambda_1}^3 \cdot (2B + A - 1) + {\lambda_2}^3 \cdot B + e$	${\lambda_1}^3$	${\lambda_1}^3 \cdot A + \lambda_1 \cdot \lvert S\rvert + {\lambda_2}^3$	$\lambda_1 \cdot \lvert S_a\rvert + 2 \cdot e$
[15, $\mathcal{S}$3.1] 3.2	$2 \cdot e + {\lambda_1}^3 + {\lambda_2}^3$	${\lambda_1}^3 \cdot N + {\lambda_2}^3$	$\lambda_1 \cdot \lvert S\rvert + {\lambda_1}^3 + {\lambda_2}^3$	$\lambda_1 \cdot (\lvert S\rvert - 1) + 2 \cdot e$
[23] 3.3	$7 \cdot {\lambda_1}^3 + 6 \cdot {\lambda_2}^3 + 2 \cdot e + 2 \cdot \lambda_1$	${\lambda_1}^3 \cdot (N + 8) + 2 \cdot {\lambda_2}^3 + 5 \cdot \lambda_1$	$\lambda_1 \cdot (\lvert S\rvert + 2) + 6 \cdot {\lambda_1}^3 + 6 \cdot {\lambda_2}^3$	$\lambda_1 \cdot (\lvert S\rvert - 1) + 9 \cdot e$

Table 10. Theoretical runtimes, where N is the total number of users in the broadcast system.

Scheme	Theoretical Runtime			
	Setup	KeyGen	Enc	Dec
baseline 2	$O(\lambda^3 \cdot N)$	—	$O(\lambda^3 \cdot \lvert S\rvert + \lambda \cdot \lvert S\rvert)$	$O(\lambda)$
[2, $\mathcal{S}$3.1] 3.1	$O(\lambda^3 \cdot N)$	$O(\lambda^3)$	$O(\lambda \cdot \lvert S\rvert)$	$O(\lambda \cdot \lvert S\rvert)$
[2, $\mathcal{S}$3.2] 3.1	$O(\lambda^3 \cdot \sqrt{N})$	$O(\lambda^3)$	$O(N + \lambda^3 \cdot \sqrt{N} + \lambda \cdot \lvert S\rvert)$	$O(\lambda \cdot \lvert S_a\rvert)$
[15, $\mathcal{S}$3.1] 3.2	$O(\delta_1 \cdot N)$	$O(\lambda^3 \cdot N)$	$O(\lambda \cdot \lvert S\rvert)$	$O(\lambda \cdot \lvert S\rvert)$
[23] 3.3	$O(\delta_1 \cdot N)$	$O(\lambda^3 \cdot N)$	$O(\lambda \cdot \lvert S\rvert)$	$O(\lambda \cdot \lvert S\rvert)$

On Identity-Based and Tracing Broadcast Encryption Systems. In the full version [4], we compare the identity-based broadcast encryption systems from [15, $\mathcal{S}$4.1], [15, $\mathcal{S}$4.3.1], and [14, $\mathcal{S}$3.1]. We also compare a wide-range of systems that support tracing, including a private-linear broadcast encryption (PLBE) system from [12], an augmented broadcast encryption (ABBE) system from [13], and a risky broadcast and trace multi-scheme from [24]. We defer these details to [4] for space reasons.

4 Applications of Broadcast Encryption

Online Video Streaming. The most commonly referenced use case for broadcast encryption is online video streaming services like Disney+, Netflix, and Hulu. This category can also include content streamed by individuals on platforms like Twitch and YouTube, online conferencing services like Zoom and Microsoft Teams and even many social media platforms like Facebook, Instagram and TikTok. 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 [6], 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 [2] 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 [2, $\mathcal{S}$3.1] variant provides the best ciphertext size at 96B per ciphertext while decryption takes 350 ms and the [2, $\mathcal{S}$3.2] variant provides the fastest decryption at 1.6 ms with a 32 KB ciphertext. Either of these are reasonable choices, although if the ciphertext size isn't a problem, we'd recommend [2, $\mathcal{S}$3.2] due to its smaller public key size (of 160 KB, where as [2, $\mathcal{S}$3.1] requires 128 MB for 1 million users). For very large user datasets (e.g., in the 190 million range), using [2, $\mathcal{S}$3.2] instead of [2, $\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 [2] schemes were proven secure in the static security model; if one wants the stronger adaptive security, Waters [23] offers this and small 861B ciphertexts, although the public key sizes grow to 32 MB for $N = 1$ million. Both of the identity-based systems [14,15] require *hours* to decrypt a single ciphertext when N is 1 million, so they are not contenders.

The performance hit from [12] (the best performing traitor tracing scheme) vs. [2] could be worth it for the chance to combat revenue reducers like piracy. For $N = 1$ million users, the decryption time of [12] doubles (over [2, $\mathcal{S}$3.2]) to 3.2 ms while the ciphertext size grows by a factor of 19 to 605 KB – larger, but reasonable on fast networks. The public key size roughly triples to 477 KB. Zhandry's risky traitor tracing scheme [24, $\mathcal{S}$9.3] provides the best ciphertext size for tracing schemes at only 38 4B, but the decryption time explodes to an infeasible 19 h (for $N = 1$ million). Zhandry's nonrisky, post-user expansion compiler version of his scheme (see [4] for details) has decryption times that are comparable to those for [12] and [13] for $N = 1$ million, but the encryption time balloons to over 1 min and the ciphertext size jumps from roughly 600 KB to almost 4 MB (details in [4] when $a = 2/3$.) One potential benefit is that the public/private key sizes of Zhandry's scheme are smaller, but that likely won't offset the additional encryption and space overhead.

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

Recommendation: Use [2, $\mathcal{S}$3.2] (for fastest decryption and scalable public key size). See Sect. 3.1. If traitor tracing is required, use [12].

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. Currently the way that online game streaming is done is that a server runs the game program remotely for each individual user. However, if adapted a multiplayer game could feasibly send the same stream out to every user maximizing both server-side and client side efficiency.

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 [20], 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 [2, $\mathcal{S}$3.2] or [23]. Both schemes have ciphertexts under 1 KB even for 1 million users, but the primary cost for both is a jump in public key size of 128 MB and 32 MB respectively. The [23] offers stronger provable security and low encryption/decryption times, while [2, $\mathcal{S}$3.2] offers decryption times that are two orders of magnitude faster but encryption time is roughly triple that of [23].

While the live traitor tracing functionality could be useful, the difference in performance could make a notable difference for users. Perhaps [12] 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 [2]. 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 s 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: Needs to scale to 1 million users or more, with a combination of ciphertext size overhead and (client side) encryption and decryption times that support live interactions. Need to balance benefits of tracing with impact on user experience.

Recommendation: Use [23] (for strong overall balance of security, low size overhead and fast encryption/decryption times) or if more speed in one component is needed, use [2, $\mathcal{S}$3.2] (for fastest decryption) or [2, $\mathcal{S}$3.1] (for fastest transmission). See Tables 7 and 8. The overhead required for traitor tracing may frustrate the live gaming experience, but if it is needed, a hybrid approach using [12] may work. See [4] for more details.

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. Additionally, some of these services may include a live broadcast, which could be different from the public broadcast (i.e. a bettors specific broadcast). 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. 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 [15] or [2, $\mathcal{S}$3.1]. For the $N \leq 100,000$ range, the public keys of [2, $\mathcal{S}$3.1] are 13 MB, while the public keys of [15] are a more tolerable 3MB. Systems [15] and [2, $\mathcal{S}$3.1] tie for the shortest ciphertexts at 96 B. The fastest encryption plus decryption time is [15] 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: Looking for a sweet spot in the 10,000 to 100,000 user range, with a combination of ciphertext size overhead and (server side) encryption and (client side) decryption times that support real-time interactions.

Recommendation: Use [15] (for best ciphertext size, best sum of encryption and decryption time, and public key size tolerable for $N \leq 100,000$). See Sects. 3.2. The overhead required for traitor tracing may frustrate the live betting experience, but if needed, a hybrid approach using [12] and [15] may work.

Distributor Limited Applications. In the above applications, we assume that the distributor (e.g., Netflix, YouTube) has large computing resources at its disposal. However, we also anticipate use cases where distributor performance becomes a bottleneck (e.g., where a person is streaming video from their smartphone to a group). A distributor limited implementation could be relevant in both the private and public sector. Within the private sector, a company manager who wants to broadcast a specific message to his employees could do so using broadcast encryption. With the presumed post-social-distancing increase in online work, consistent, secure communication between manager and employees could be increasingly important with a decline in face-to-face communication.

Within the public and military sector, these same benefits apply. In the public sector, however, having differing levels of access and the ability to revoke access to messages and live communications is more important. For example, if a broadcast encryption system is used to send out orders to a group, and one of the recipient devices is captured then revocation is necessary. Additionally, traitor tracing functionality could be especially valuable.

Thus, in the case of a direct peer-to-many-peer type of communication, the performance of the distributor system becomes relevant, thus making the times for the Setup, KeyGen and Encrypt function times more critical. In this situation, a simple recommendation is harder to make. [2, $\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 [2, $\mathcal{S}$3.2] is much worse than [2, $\mathcal{S}$3.1] and [15, $\mathcal{S}$3.1]. Due to that constraint, if there are limited resources for the distributor, using [2, $\mathcal{S}$3.2] isn't a good choice. If peers are less than 100 K, in a situation with a distributor bottleneck we'd recommend either [2, $\mathcal{S}$3.1] or [15, $\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, [2, $\mathcal{S}$3.1] would be preferable, but in cases where keys are generated less often [15, $\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 1,000, [13] slightly outperforms [12] 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, [12] will perform better. In a situation where keys are generated less often and there are less than 1,000 users, [13] can perform better. However, both of these schemes are outperformed by the baseline (individual encryption to each peer) until about about the 10,000 user level.

Constraint Summary: For $N \leq 100,000$ user range, looking to optimize the distributor functions without sacrificing much data transfer time or client performance.

Recommendation: Use [2, $\mathcal{S}$3.1] (if need to generate keys often) or [15, $\mathcal{S}$3.1] (otherwise). See Tables 7 and 8. If traitor tracing is required for under 10,000 users, the baseline (individual encryption) will likely outperform any of the tracing broadcast schemes. If traitor tracing is required for over 10,000 users, use [12]. See [4] for more details.

Acknowledgments. The authors are grateful to Mark Zhandry for helpful interactions regarding his work [24] and Brent Waters for helpful discussions regarding prior work in broadcast encryption.

References

1. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: IEEE Symposium on Security and Privacy, pp. 321–334 (2007)
2. Boneh, D., Gentry, C., Waters, B.: Collusion resistant broadcast encryption with short ciphertexts and private keys. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 258–275. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_16
3. Boneh, D., Sahai, A., Waters, B.: Fully collusion resistant traitor tracing with short ciphertexts and private keys. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 573–592. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_34
4. Chhatrapati, A., Hohenberger, S., Trombo, J., Vusirikala, S.: A performance evaluation of pairing-based broadcast encryption systems. Cryptology ePrint Archive, Report 2021/1526 (2021). https://ia.cr/2021/1526

5. Company, T.W.D.: Fiscal year 2019 annual financial report (2019). https://thewaltdisneycompany.com/app/uploads/2020/01/2019-Annual-Report.pdf
6. Corporation, N.: Netflix Q2 2020 shareholder letter, 16 July 2020. https://s22.q4cdn.com/959853165/files/doc_financials/2020/q2/FINAL-Q2-20-Shareholder-Letter-V3-with-Tables.pdf
7. Delerablée, C.: Identity-based broadcast encryption with constant size ciphertexts and private keys. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 200–215. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76900-2_12
8. Dolev, D., Dwork, C., Naor, M.: Nonmalleable cryptography. SIAM J. Comput. **30**(2), 391–437 (2000)
9. Fiat, A., Naor, M.: Broadcast encryption. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 480–491. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48329-2_40
10. Fujisaki, E., Okamoto, T.: How to enhance the security of public-key encryption at minimum cost. In: Imai, H., Zheng, Y. (eds.) PKC 1999. LNCS, vol. 1560, pp. 53–68. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-49162-7_5
11. Gamal, T.E.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inf. Theory **31**(4), 469–472 (1985). https://doi.org/10.1109/TIT.1985.1057074
12. Garg, S., Kumarasubramanian, A., Sahai, A., Waters, B.: Building efficient fully collusion-resilient traitor tracing and revocation schemes. IACR Cryptol. ePrint Arch. **2009**, 532 (2009). http://eprint.iacr.org/2009/532
13. Garg, S., Kumarasubramanian, A., Sahai, A., Waters, B.: Building efficient fully collusion-resilient traitor tracing and revocation schemes. In: ACM CCS, pp. 121–130. ACM (2010). https://doi.org/10.1145/1866307.1866322
14. Ge, A., Wei, P.: Identity-based broadcast encryption with efficient revocation. In: Lin, D., Sako, K. (eds.) PKC 2019. LNCS, vol. 11442, pp. 405–435. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17253-4_14
15. Gentry, C., Waters, B.: Adaptive security in broadcast encryption systems (with short ciphertexts). In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 171–188. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_10
16. Goldwasser, S., Micali, S.: Probabilistic encryption. J. Comput. Syst. Sci. **28**(2), 270–299 (1984)
17. Goyal, R., Koppula, V., Russell, A., Waters, B.: Risky traitor tracing and new differential privacy negative results. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 467–497. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_16
18. Goyal, R., Koppula, V., Waters, B.: Lockable obfuscation. In: FOCS, pp. 612–621 (2017)
19. Naor, M., Yung, M.: Public-key cryptosystems provably secure against chosen ciphertext attacks. In: STOC, pp. 427–437 (1990)
20. Nvidia: 2020 Nvidia Corporation Annual Review (2020). https://s22.q4cdn.com/364334381/files/doc_financials/2020/ar/2020-nvidia-annualreport-content-r25-web-144dpi-combined.pdf
21. Rackoff, C., Simon, D.R.: Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 433–444. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_35

22. Sakai, R., Furukawa, J.: Identity-based broadcast encryption. IACR Cryptol. ePrint Arch. (2007). http://eprint.iacr.org/2007/217
23. Waters, B.: Dual system encryption: realizing fully secure IBE and HIBE under simple assumptions. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 619–636. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03356-8_36
24. Zhandry, M.: New techniques for traitor tracing: Size $N^{1/3}$ and more from pairings. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12170, pp. 652–682. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56784-2_22