Collaborative, Privacy-Preserving Data Aggregation at Scale

Benny Applebaum*, Haakon Ringberg†, Michael J. Freedman†, Matthew Caesar°, Jennifer Rexford†

*Weizmann Institute, †Princeton University, °UIUC

Abstract. Combining and analyzing data collected at multiple administrative locations is critical for a wide variety of applications, such as detecting malicious attacks or computing an accurate estimate of the popularity of Web sites. However, legitimate concerns about privacy often inhibit participation in collaborative data aggregation. In this paper, we design, implement, and evaluate a practical solution for privacy-preserving data aggregation (PDA) among a large number of participants. Scalability and efficiency is achieved through a "semi-centralized" architecture that divides responsibility between a *proxy* that obliviously blinds the client inputs and a *database* that aggregates values by (blinded) keywords and identifies those keywords whose values satisfy some evaluation function. Our solution leverages a novel cryptographic protocol that provably protects the privacy of both the participants and the keywords, provided that proxy and database do not collude, even if both parties may be individually malicious. Our prototype implementation can handle over a million suspect IP addresses per hour when deployed across only two quad-core servers, and its throughput scales linearly with additional computational resources.

1 Introduction

Many important data-analysis applications must aggregate data collected by multiple participants. ISPs and enterprise networks may seek to share traffic mix information to more accurately detect and localize anomalies. Similarly, collaboration can help identify popular Web content by having Web users—or proxies monitoring traffic for an entire organization—combine their access logs to determine the most frequently accessed URLs [1]. Such distributed data analysis is similarly important in the context of security. For example, victims of denial-of-service (DoS) attacks know they have been attacked but cannot easily distinguish the malicious source IP addresses from the good users who happened to send legitimate requests at the same time. Since compromised hosts in a botnet often participate in multiple such attacks, victims could potentially identify the bad IP addresses if they combined their measurement data [39]. Cooperation is also useful for Web clients to recognize they have received a bogus DNS response or a forged self-signed certificate, by checking that the information they received agrees with that seen by other clients accessing the same Web site [34, 41]. In this paper, we present the design, implementation, and evaluation of an efficient, privacy-preserving system that supports these kinds of data analysis.

Today, these kinds of distributed data aggregation and analysis lack privacy protections. Existing solutions often rely on a trusted (typically centralized) aggregation node that collects and analyzes the raw data, thereby learning both the identity and inputs of participants. There is good reason to believe this inhibits participation. ISPs and

Web sites are notoriously unwilling to share operational data with one another, because they are business competitors and are concerned about compromising the privacy of their customers. Many users are unwilling to install software from Web analytics services such as Alexa [1], as such software would track and report every Web site they visit. Unfortunately, even good intentions may not translate to good privacy protections, demonstrated all too well by the fact that large-scale data breaches have become commonplace [35]. There certainly are non-Internet applications as well. Patients could benefit from the aggregated analysis of medical data, but there are significant privacy concerns—and regulation in the form of HIPAA and laws—that prevent such. As such, we believe that many useful distributed data-analysis applications will not gain serious traction unless privacy can be ensured.

Fortunately, many of these collaborative applications have a common pattern: aggregating participants' inputs on common input keys and potentially analyzing the resulting intersection. When designed with privacy in mind, we refer to this problem as *privacy-preserving data aggregation* (PDA). Namely, each participant p_j (or *client*) autonomously makes observations about *values* associated with *keys*, *i.e.*, input key-value tuples $\langle k_i, v_i \rangle$. The system jointly computes a two-column input table T. The first column of T is a set comprised of all unique keys belonging to all participants (the *key column*). The second, *value column* is comprised of values $\mathsf{T}[k_i]$ that are the sum or union of all participant's values for k_i. This is akin to a database join on matching keys across each participant's input (multi)set.

We consider two different forms of this functionality: (1) *aggregation-only (PDA)*, where the output is just the value column, and (2) *conditional-release (CR-PDA)*, where the protocol also outputs a key k_i if and only if some evaluation function $f(\forall j | v_{i,j})$ is satisfied. For example, our botnet anomaly detection is an instance of over-threshold set intersection—also known as the heavy-hitter or iceberg detection problem—where the goal is to detect keys that occur more than some threshold number of times across all participants. Here, the keys k_i refer to IP addresses, each value $v_{i,j}$ is 1, and f is true iff its cardinality exceeds some threshold τ (*i.e.*, if values are aggregated as $\mathsf{T}[k_i] \leftarrow \mathsf{T}[k_i] + 1$, is $\mathsf{T}[k_i] \geq \tau$?)

A practical PDA system should provide the following:

- ***Keyword privacy:*** No party should learn anything about inputted keys. That is, given the above aggregated table T, each party should only learn the value column $\mathsf{T}[k_i]$ at the conclusion of the protocol. In the case of CR-PDA, parties should only learn the keys k_i whose corresponding value $\mathsf{T}[k_i]$ satisfies f.
- ***Participant privacy:*** No party should learn which key inputs belongs to which participant (except for information which is trivially deduced from the output of the function). This is formally captured by showing that the protocol leaks no more information than an ideal implementation that uses a trusted third party, a convention standard in secure multi-party computation [19].
- ***Efficiency:*** The system should scale to large numbers of participants, each generating and inputting large numbers of observations (key-value tuples). The system should be scalable both in terms of the bandwidth consumed (communication complexity) and the computational complexity of executing the PDA.
- ***Flexibility:*** There are a variety of computations one might wish to perform over each key's values $\mathsf{T}[k_i]$, other than the sum-over-threshold test. These may include

Approach	Keyword Privacy	Participant Privacy	Efficiency	Flexibility	Lack of Coordination
Garbled-Circuit Evaluation [42, 3]	**Yes**	**Yes**	Very Poor	**Yes**	No
Multiparty Set Intersection [16, 26]	**Yes**	**Yes**	Poor	No	No
Hashing Inputs [17, 2]	No	No	**Very Good**	**Yes**	**Yes**
Network Anonymization [11]	No	**Yes**	**Very Good**	**Yes**	**Yes**
This paper	**Yes**	**Yes**	**Good**	**Yes**	**Yes**

Table 1: Comparison of proposed schemes for privacy-preserving data aggregation

finding the maximum value for a given key, or checking if the median of a row exceeds a threshold. A single protocol should work for a wide range of functions.

– ***Lack of coordination:*** Finally, the system should operate without requiring that all participants coordinate their efforts to jointly execute some protocol at the same time, or even all be online around the same time. Furthermore, no set of participants should be able to prevent others from executing the protocol.

Classes of solutions. Privacy-preserving data aggregation is a form of the general *secure multiparty computation* problem where multiple participants wish to jointly compute some value based on individually-held secret bits of information without revealing their secrets to one another. The theoretical cryptographic literature provides generic solutions for this problem which also satisfy very strong notions of security [42, 20, 4, 7]. In general, however, these tools are not efficient enough to be used in practice. Few have ever been implemented ([28, 18, 3]), let alone operated in the real world [5]. Moreover, they do not scale well either to large data sets or to a large number of participants. More efficient solutions exist for special cases of the PDA problem, such as secure set intersection [13, 30, 27, 16, 26, 15, 23, 10]. However, while some of these solutions are quite efficient when the number of participants is small (e.g., 2), none of them achieve practical efficiency in our setting where there are hundreds or thousands of participants each generating thousands of inputs.[1]

On the other extreme, ad-hoc solutions for PDA can be highly efficient. Rather than building fully decentralized protocols, we could aggregate data and compute results using a centralized server. One approach is to simply have clients first hash their keys before submitting them to the server (*e.g.*, using SHA-256), so that a server only sees $H(k_i)$ [2]. While it may be difficult to find the hash function's pre-image, brute-force attacks may be possible. In our intrusion detection application, for instance, a server can easily compute the hash values of all four billion IP addresses and build a simple lookup table. Thus, while efficient, this approach fails to achieve either keyword or participant privacy, with the latter not achieved because a client submits its inputs directly to the server. That said, one possible approach for participant privacy would be to proxy a client's request through one or more intermediate proxies that hide the client's identity (*e.g.*, its IP address), as done in network anonymity systems such as Tor [11].

Table 1 summarizes these design points. An important goal of this work is to provide a solution between these two extremes, *i.e.*, a protocol that is efficient enough to be used in practice and at large scale, yet also provide a meaningful level of security that is

[1] For example, a careful protocol implementation of [16] found two sets of 100 items each took 213 seconds to execute [18].

formally provable. There are various ways one could imagine weakening the strongest notions of secure multi-party computation, which provide privacy guarantees against *any* malicious participant. A standard relaxation would be to only guarantee privacy against *honest-but-curious* parties, in which participants learn no information provided that they faithfully execute the correct protocol. Another approach would be to provide privacy against all small coalitions of malicious parties. But in the large settings we consider, it may be easy for a single party to forge multiple identities and thus circumvent such protections, the so-called Sybil attack [12].

Instead, we focus on providing security against any malicious participant, provided that there exists a small set of well-known parties that do not collude. This is a natural model that already appears in real-world scenarios, such as Democrats and Republicans jointly comprising election boards in the U.S. political system. For our specific examples, business competitor ISPs like AT&T and Sprint could jointly provide a service like cooperative DoS detection. Or, it could be offered by third-party entities who have no incentive to collude. Such non-collusion assumptions already appear in several cryptographic protocols [8, 14]. It should be emphasized that these well-known parties should not be treated as trusted: we only assume that they will not collude. Indeed, jumping ahead, our protocols do not reveal sensitive information to either party.

Contributions. In this paper, we *design, implement, and evaluate* privacy-preserving data aggregation—through logical centralization over a small number of non-colluding parties—that provably offers privacy-preserving data aggregation without sacrificing efficiency. Rather than full decentralization (as in secure multi-party computation) or full centralization (as typical in trusted-party solutions), our PDA architecture is split between well-known entities playing two different roles: a *proxy* and a *database* (DB). The proxy plays the role of obliviously blinding client inputs, as well as transmitting blinded inputs to the DB. The DB, on the other hand, builds a table that is indexed by the blinded key and aggregates each row's values (either incrementally or after some time). While most of the paper will focus on the case of only two entities—one proxy and one DB—we also show how to extend the protocol to larger numbers of parties.

The resulting system provides strong keyword and participant privacy guarantees, provided that the well-known entities—which operate the proxy and the database—do not collude. Specifically, we describe two variants of the protocol which provides the following notions of security (see Appendix A for more details):

- ***Privacy of PDA against malicious entities and malicious participants***: Even an arbitrary coalition of malicious participants, together with either a malicious proxy or DB, learn nothing about other participants' inputs (except that implied by the protocols' output). Such a coalition may violate correctness in almost arbitrary ways, however. Similar notions of security have appeared before [32, 15, 23].
- ***Privacy of CR-PDA against honest-but-curious entities and malicious participants***: Our CR-PDA protocol achieves full security in the "ideal-real" framework. This holds with respect to malicious coalitions of participants, as well as honest-but-curious coalitions between participants and the DB or proxy.

Using a semi-centralized architecture greatly reduces operational complexity and simplifies the liveness assumptions of the system. Clients can asynchronously provide

inputs without our system requiring any complex scheduling. Despite these simplifications, the cryptographic protocols necessary to provide strong privacy guarantees are still non-trivial. Specifically, our solution makes use of oblivious pseudorandom functions [33, 15, 23], amortized oblivious transfer [31, 24], and homomorphic encryption with re-randomization. In summary, the contributions of this paper include:

- We demonstrate a tradeoff between efficiency and security in multi-party computation. Our protocols achieve a relatively strong notion of provable security, while are still practical for large numbers of participants with large input sets.
- At an abstract level, we introduce and implement a new cryptographic primitive that extends the notion of oblivious pseudorandom function (OPRF) as follows: A sender with input k communicates with a receiver *via* a mediator who holds a PRF key s. At the end of the protocol, the receiver learns $F_s(k)$, and the sender and mediator learn nothing. We believe that this notion, as well as our specific implementation, are of independent cryptographic interest and may be useful elsewhere.
- There are very few implementations of secure multi-party computation ([28, 3, 5]), and our system is one of the first to demonstrate practical efficiency. To our knowledge, it also includes the first implementation of some cryptographic machinery we use as sub-protocols (*e.g.*, amortized oblivious transfer [24]); our evaluation show that they realize significant benefits in practice.
- Finally, we illustrate that our system provides a level of performance that is sufficient for several applications of interest, including anomaly detection, certificate cross-checking, and distributed ranking.

The remainder of this paper is organized as follows. Section § 2 describes our PDA protocols and sketches proofs of their privacy. We describe our system architecture and implementation in §3, evaluate its performance in §4, and conclude in §5. The appendix details some security definitions, protocol extensions, and proofs.

2 Our Protocols

In this section, we describe our protocols and analyze their security. We first describe a simplified version of the CR-PDA protocol that achieves somewhat weaker security properties, and we then extend this protocol to support a stronger notion of security. More details about the extended protocol and sketches of formal security proofs are given in the appendix. We conclude by explaining how to adopt the CR-PDA protocol to support the (simpler) case of the PDA functionality and sketch an extension to the case of $t > 2$ mutually-distrustful parties.

2.1 The Basic CR-PDA Protocol

Our protocol consists of five basic steps (see Figure 1). In the first two steps, the proxy interacts with the participants to collect the blinded keys together with their associated values encrypted under the DB's public key, and then passes these encrypted values on to the DB. Then, in the next two steps, the DB aggregates the blinded keys with the associated values in a table, and it decides which rows should be revealed according to a predefined function f. Finally, the DB asks the proxy to unblind the corresponding keys. Since the blinding scheme F_s is not necessarily invertible, the revealing mechanism uses additional information sent during the first phase. The specific steps are as follows.

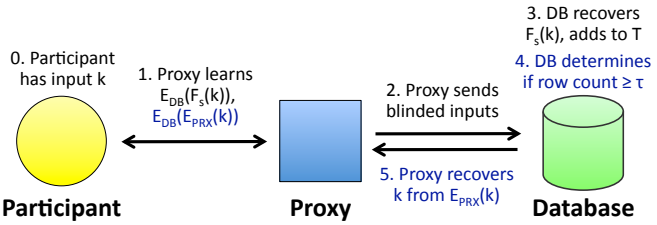


Fig. 1: High-level system architecture and protocol. Conditional release extensions to PDA are steps 4 and 5, as well as additional input in step 2 (all shown in blue). F_s **is a keyed hash function whose secret** s **is known only to the proxy.**

– **Parties**: Participants, Proxy, Database.
– **Cryptographic Primitives**: A pseudorandom function F, where $F_s(k_i)$ denotes the value of the function on input k_i with a key s. A public-key encryption E, where $E_{\mathrm{K}}(x)$ denotes an encryption of x under the public key K.
– **Public Inputs**: The proxy's public key PRX, the database's public key DB.
– **Private Inputs.** *Participant:* A list of key-value pairs $\langle k_i, v_i \rangle$. *Proxy:* key s of PRF F and secret key for PRX; *Database:* secret key for DB.

1. Each participant interacts with the proxy as follows. For each entry $\langle k_i, v_i \rangle$ in its list, the participant and the proxy run a sub-protocol for oblivious evaluation of the PRF (OPRF). At the end of this sub-protocol, the proxy learns nothing and the participant learns only the value $F_s(k_i)$ (and nothing else, not even s). The participant computes $E_{\mathrm{DB}}(F_s(k_i))$, $E_{\mathrm{DB}}(v_i)$, and $E_{\mathrm{DB}}(E_{\mathrm{PRX}}(k_i))$, and it sends them to the proxy. (The last entry will be used during the revealing phase.) The proxy adds this triple to a list and waits until most/all participants send their inputs.
2. The proxy randomly permutes the list of triples and sends the result to the DB.
3. The DB decrypts all the entries of each triple. Now, it holds a list of triples of the form $\left\langle F_s(k_i), v_i, E_{\mathrm{PRX}}(k_i) \right\rangle$. If a value v_i is not valid (*i.e.*, $v_i \notin \mathcal{D}$), the corresponding triple is omitted. The DB inserts the valid values into a table which is indexed by the blinded key $F_s(k_i)$. At the end, the DB has a table of entries of the form $\left\langle F_s(k_i), \mathsf{T}[k_i], \mathsf{E}[k_i] \right\rangle$. $\mathsf{T}[k_i]$ is some aggregation of all v_i's that appeared with k_i (*e.g.*, the actual values or, for threshold set intersection, simply the number of times a that k_i was inputted). $\mathsf{E}[k_i]$ is a list of values of the form $E_{\mathrm{PRX}}(k)$.
4. The DB uses some predefined function f to partition the table into two parts: R, which consists of the rows whose keys should be revealed, and H, which consists of the rows whose keys should remain hidden. It sends R to the proxy.
5. The proxy goes over the received table R and replaces all the encrypted $E_{\mathrm{PRX}}(k_i)$ entries with their decrypted key k_i. It then publishes the updated table.

Security Guarantees. This protocol guarantees privacy against the following:

Coalition of honest-but-curious (HBC) participants. Consider the view of an HBC participant during the protocol. Due to the security of the OPRF, a single participant sees only a list of pseudorandom values $F_s(k_i)$, and therefore this view can be easily simulated by using truly random values. The same holds for any coalition of participants. In fact, this protocol achieves reasonable security against malicious participants

as well. The interaction of the proxy with a participant is completely *independent* of the inputs of other participants. Hence, even if participants are malicious, they still learn nothing about other participants' inputs. Furthermore, even malicious participants will be forced to choose their inputs *independently* of other honest participants. (See [31, 23] for similar security definitions.) However, malicious participants can still violate the *correctness* of the above protocol. We fix this issue in the extended protocol.

HBC coalition of proxy and participants. The proxy's view consists of three parts: (1) the view during the execution of the OPRF protocol—this gives no information due to the security of the OPRF; (2) the tuples that the participants send—these values are encrypted under the DB's key and therefore reveal no information to the proxy; and (3) the values that the DB sends during the last stage of the protocol—these are just key-value pairs (encrypted under the proxy's key) that should be revealed anyway, and thus they give no additional information beyond the actual output of the protocol.

This argument generalizes to the case where the proxy colludes with HBC participants: their joint view reveals nothing about the inputs of the honest participants.

HBC database. The DB sees a blinded list of keys encrypted under his public key DB, without being able to relate blinded entries to their owners. For each blinded key $F_s(k_i)$, the DB sees the list of its associated values $\mathsf{T}[k_i]$ and encryptions of the keys under the proxy's key PRX. Finally, the DB also sees the protocol's output. The values $F_s(k_i)$ and $E_{\text{PRX}}(k)$ bear no information due to the security of the PRF and the encryption scheme. Hence, the DB learns nothing but the value table of the inputs, *i.e.*, the $\mathsf{T}[k_i]$'s for all k_i's. (Formally, we define a functionality in which this additional information is given to the DB as part of its output; see appendix.)

2.2 A More Robust PDA Protocol

We now describe how to immunize the basic protocol against stronger attacks.

HBC coalition of participants and DB. The previous protocol is vulnerable against such coalitions for two main reasons.

First, a participant knows the blinded version $F_s(k_i)$ of its own keys k_i, and, in addition, the DB can associate all the values $\mathsf{T}[k_i]$ to their blinded keys $F_s(k_i)$. Hence, a coalition of a participant and a DB can retrieve all the values $\mathsf{T}[k_i]$ that are associated with a key k_i that the participant holds, even if this key *should not be revealed* according to f. To fix this problem, we modify the first step of the protocol. Instead of using an OPRF protocol, we will use a different sub-protocol in which the participant learns nothing and the proxy learns the value $E_{\text{DB}}(F_s(k_i))$ for each k_i. This solves the problem as now that participant himself does not know the blinded version of his own keys. To the best of our knowledge, this version of an *encrypted-OPRF protocol* (abbreviated EOPRF and detailed in §2.3) has not previously appeared in the literature.

Second, we should eliminate subliminal channels, as these can be used by participants and the DB to match the keys of a participant to their blinded versions. To solve this problem, we use an encryption scheme that supports re-randomization of ciphertexts; that is, given an encryption of x with randomness b, it should be possible to recompute an encryption of y under fresh randomness b' (without knowing the private key). Now we eliminate the subliminal channel by asking the proxy to re-randomize the

ciphertexts—$E_{\mathrm{DB}}(F_s(k_i))$, $E_{\mathrm{DB}}(v_i)$, and $E_{\mathrm{DB}}(E_{\mathrm{PRX}}(k_i))$—which are encrypted under the DB's public key (at Step 1). We should also be able to re-randomize the *internal* ciphertext $E_{\mathrm{PRX}}(k_i)$ of the last entry as well.

Coalition of malicious participants. As we observed, malicious participants can violate the correctness of our protocol, *e.g.*, by trying to submit ill-formed inputs. Recall that the participant are supposed to send to the proxy triples $\langle a, b, c\rangle$, of the form $a = E_{\mathrm{DB}}(F_s(k_i)), b = E_{\mathrm{DB}}(v_i)$ and $c = E_{\mathrm{DB}}(E_{\mathrm{PRX}}(k_i))$ for some k_i and v_i. However, a cheating participant might provide an inconsistent tuple, in which $a = E_{\mathrm{DB}}(F_s(k_i))$ while $c = E_{\mathrm{DB}}(E_{\mathrm{PRX}}(k'_i))$ for some $k'_i \neq k_i$. To prevent this attack, we let the proxy apply a consistency check to R in the last step of the protocol. The proxy makes sure that $E_{\mathrm{PRX}}(k'_i)$ and $F_s(k_i)$ match, and otherwise omits the inconsistent values. Then the DB checks again if the corresponding row should still be revealed.(See appendix for more details.)

A cheating participant might also try to replace b with some "garbage" value $b' = E_{\mathrm{DB}}(v')$ which is not part of the legal domain $\mathcal{D}$ or for which he does not know the plaintext v'. (While this might not seem beneficial in practice, we must prevent such an attack to meet strong definitions of security.) To prevent such attacks, we use an encryption scheme which supports only messages taken from the domain $\mathcal{D}$, and ask the participant to provide a zero-knowledge proof of knowledge (ZK-POK) that he knows the plaintext v to which b decrypts. As seen later, this does not add too much overhead.

Finally, our sub-protocol for the EOPRF should be secure against malicious participants in the following sense: a malicious participant should not be able to generate a blinded value $E_{\mathrm{DB}}(F_s(k_i))$ for a key k_i that he does not know. In the appendix, we show that our modifications guarantee full security against malicious participants.

2.3 Concrete Instantiation of the Cryptographic Primitives

In the following section, we assume that the input keys are represented by m-bit strings. We assume that m is not very large (*e.g.*, less than 192–256); otherwise, one can hash the input keys and apply the protocol to resulting hashed values.

Public Parameters. We mostly employ Discrete-Log-based schemes. In the following, g is a generator of a multiplicative group $\mathbb{G}$ of prime order p for which the decisional Diffie-Hellman assumption holds. We publish (g, p) during initialization and assume that algorithms for multiplication (and thus for exponentiation) in $\mathbb{G}$ exist.

El-Gamal Encryption. We will use El-Gamal encryption over the group $\mathbb{G}$. The private key is a random element a from $\mathbb{Z}_p^*$, and the public key is the pair $(g, h = g^a)$. In case we wish to "double-encrypt" a message $x \in \mathbb{G}$ under two different public keys (g, h) and (g, h'), we will choose a random b from $\mathbb{Z}_p^*$ and compute $(g^b, x \cdot (h \cdot h')^b)$. This ciphertext as well as standard ciphertexts can be re-randomized by multiplying the first entry (resp. second entry) by $g^{b'}$ (resp. $h^{b'}$), where b' is chosen randomly from $\mathbb{Z}_p^*$.

Goldwasser-Micali Encryption. The values v_i which are taken from the domain $\mathcal{D}$ will be encrypted under the Goldwasser-Micali (GM) Encryption scheme [21]. Specifically, if the domain size is 2^ℓ, we represent the values of $\mathcal{D}$ by all possible ℓ-bit strings, and encrypt such strings under GM in a bit-by-bit manner. The GM scheme provides ciphertext re-randomization, and it allows the party who generates a ciphertext c to prove in zero-knowledge that he knows the decryption of c and that c is valid (*i.e.*, decrypts

to an ℓ bit string) [22]. Furthermore, both these operations and encryption cost only ℓ modular multiplications.[2] Decryption costs 2ℓ modular exponentiations, but ℓ is typically bounded by a very small integer in our protocols. Finally, the ZK proof consists of 3 moves and can run in parallel with the EOPRF.

Naor-Reingold PRF [33]. The key s of the function $F_s : \{0,1\}^m \to \mathbb{G}$ contains m values $(s_1, \ldots, s_m)$ chosen randomly from $\mathbb{Z}_p^*$. Given m-bit string $k = x_1 \ldots x_m$, the value of $F_s(k)$ is $g^{\prod_{x_i=1} s_i}$, where the exponentiation is computed in the group $\mathbb{G}$.

Oblivious-Transfer [36, 31] and Batched Oblivious Transfer [24]. To implement the sub protocol of Step 1, we need an additional cryptographic tool called Oblivious Transfer (OT). In an OT protocol A sender holds two strings (α, β), and a receiver has a selection bit c. At the end of the protocol, the receiver learns a *single* string: α if $c = 0$, and β if $c = 1$. The sender learns nothing (in particular, it does not learn c). In general, OT is an expensive public-key operation (*e.g.*, it may take two exponentiations per invocation and, in the above protocol, we would execute OT for each *bit* of the participant's input k_i). However, Ishai *et al.* [24] show how to reduce the amortized cost of OT to be as fast as matrix multiplication. This "batch OT" protocol uses a standard OT protocol as a building block; we built our batch OT on top of [31].

2.4 The Encrypted-OPRF protocol

Our construction is inspired by a protocol for oblivious evaluation of the PRF F [15, 30, 31]. We believe that this construction might have further applications.

- **Parties**: Participant, Proxy.
- **Inputs.** *Participant:* m-bit string $k = (x_1 \ldots x_m)$; *Proxy:* secret key $s = (s_1, \ldots, s_m)$ of a Naor-Reingold PRF F.

1. Proxy chooses m random values $u_1, \ldots, u_m$ from $\mathbb{Z}_p^*$ and an additional random $r \in \mathbb{Z}_p^*$. In parallel, for each $1 \le i \le m$: the proxy and the participant invoke the OT protocol where proxy is the sender with inputs $(u_i, s_i \cdot u_i)$ and receiver uses x_i as his selector bit. (*i.e.*, , the participant learns u_i if $x_i = 0$, and $s_i \cdot u_i$ otherwise.) The proxy also sends the value $\hat{g} = g^{r/\Pi u_i}$.
2. The participant computes the product M the values received in the OT stage. Then it computes $\hat{g}^M = (g^{\Pi_{x_i=1} s_i})^r = F_s(k)^r$, encrypts $F_s(k)^r$ under the DB's public key $\text{DB} = (g, h)$, and sends the result $(g^a, F_s(k)^r \cdot h^a)$ to the proxy.
3. The proxy raises the received pair to the power of r', where r' is the multiplicative inverse of r modulo p. It also re-randomizes the resulting ciphertext.

Correctness. Since $\mathbb{G}$ has a prime order p, the pair $(g^a, F_s(x)^r \cdot h^a)$ raised to the power of $r' = r^{-1}$, results in $(g^{ar'}, F_s(k) \cdot h^{ar'})$, which is exactly $E_{\text{DB}}(F_s(k))$.

Privacy. All the proxy sees is the random tuple $(u_1, \ldots, u_m, r)$ and $E_{\text{DB}}(F_s(k)^r)$. This view gives no additional information except of $E_{\text{DB}}(F_s(k))$. The participant, on the other hand, sees the vector $(s_1^{x_1} \cdot u_1, \ldots, s_m^{x_m} \cdot u_m)$, whose entries are randomly

[2] For the case of zero-knowledge, the protocol of [22] provides only weak soundness at the cost of ℓ multiplications. However, [9] provides strong soundness guarantees with amortized cost of ℓ modular multiplications. Our setting naturally allows such an amortization.

distributed over $\mathbb{G}$, as well as the value $\hat{g} = (g^{1/\Pi u_i})^r$. Since r is randomly and independently chosen from $\mathbb{Z}_p^*$, and since $\mathbb{G}$ has a prime order p, the element $\hat{g}$ is also uniformly and independently distributed over $\mathbb{G}$. Hence, the participant learns nothing but a sequence of random values. The protocol supports security against malicious participants (in the sense that was described earlier) and malicious proxy as long as the underlying OT is secure in the malicious setting.

2.5 Efficiency of our Protocol

In both the basic and extended protocol, the round complexity is constant, and the communication complexity is linear in the number of items. The protocol's computational complexity is dominated by cryptographic operations. For each m-bit input key, we have the following amortized complexity: (1) The participant who holds the input key computes 3 exponentiations in the basic protocol (respectively 8 in the extended protocol), as well as $O(m)$ modular multiplication / symmetric-key operations in both versions. (2) The proxy computes 5 exponentiations in the basic protocol (12 in the extended one) and $O(m)$ modular multiplication / symmetric-key operations. (3) The DB computes 3 exponentiations in the basic protocol, and $4 + \lg|\mathcal{D}|$ in the extended one where $\mathcal{D}$ is the domain of legal values. (These numbers assume that we employ RSA instead of El-Gamal whenever we can.)

2.6 Extensions and variations

PDA Protocol. Our PDA protocol is based on the CR-PDA protocol. The proxy and participant first use an EOPRF to send the proxy a list of pairs $E_{\mathrm{DB}}(F_s(k_i))$ and $E_{\mathrm{DB}}(v_i)$. (The value $E_{\mathrm{DB}}(E_{\mathrm{PRX}}(k_i))$ is not needed in this case.) Then, the proxy passes the (randomly shuffled) list to the DB, which aggregates the tuples according to the blinded keys in the table $\left\langle F_s(k_i), \mathsf{T}[k_i] \right\rangle$ and outputs the tuples $\mathsf{T}[k_i]$ in a random order. Security analysis (details omitted) is similar to the previous: malicious behavior of either proxy or DB does not affect its own view or that of a colluding participant.

Using many mutually-distrustful servers. One might want a generalized protocol with $t > 2$ servers, in which privacy holds as long as *not all* of the servers collude. We now sketch one such simple extension of our PDA protocol which works for HBC servers. This change increases the complexity by a multiplicative factor of t, and so we get a smooth tradeoff between security and efficiency.

The basic idea is to make sure that both the key of the PRF (s) and the public key of the database (DB) remain hidden from any coalition of $t-1$ servers. Specifically, each server holds a random share of an El-Gamal private key for DB (*i.e.*, the sum of the shares equals to the private key), and a key s_i for the Naor-Reingold PRF. We define a PRF $F_s(x)$ to be the product of $F_{s_1}(x), \ldots, F_{s_t}(x)$. The protocol proceeds as follows: (1) For each input $\langle k, v\rangle$, each participant performs the first EOPRF step of the previous PDA protocol with all the servers, and broadcasts the value $E_{\mathrm{DB}}(v)$. Thus, the i-th server learns the ciphertexts $\langle E_{\mathrm{DB}}(F_{s_i}(k)), E_{\mathrm{DB}}(v)\rangle$. In addition, the participant supplies to each server a POK for knowing a corresponding legal value v. (Some overhead can be saved here by using a *single* invocation of non-interactive ZK-POK.) (2) Now, the servers use the homomorphism properties of El-Gamal to compute

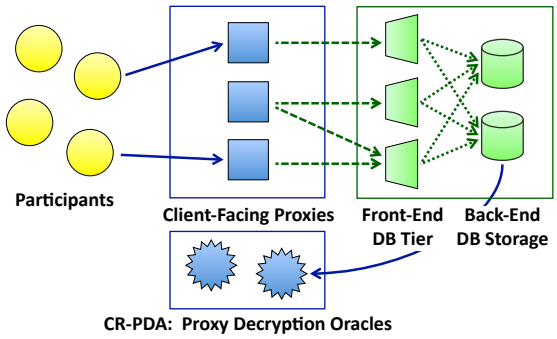


Fig. 2: Distributed proxy and database architecture

$E_{\mathrm{DB}}(F_s(k))$; they can pass the $E_{\mathrm{DB}}(F_{s_i}(k))$'s to each other in a chain-like order or via a broadcast. (3) Then, the servers emulate the second step of the previous protocol to get a randomly-ordered list of decrypted pairs $\langle F_s(k), v\rangle$. This is done in t rounds: At the i-th round, the i-th server decrypts each pair under his share of the private key (removes the i-th "layer" of encryption), rerandomizes the encryption, shuffles the list in a random order, and passes the result to the next server. The final server aggregates the values according to the blinded keys and broadcasts the result.

3 Distributed Implementation

This section describes our design and implementation of a scalable PDA architecture. For simplicity, we present the case of two administrative entities, one running a single logical proxy and the other a database. Both of these proxy and database logical components can be physically replicated in a relatively straightforward manner, however. In particular, our design can scale out horizontally to handle higher loads, by increasing the number of proxy and/or database replicas, and then distributing requests across these replicas. (Note that this replication strategy differs from the extension for $t > 2$ administrative entities, per Section 2.6.) Our distributed architecture is shown in Figure 2. Our current implementation covers all details described in the basic protocol, as well as some security improvements of the extended version (*e.g.*, including the EOPRF, but not ciphertext re-randomization, proofs of knowledge, or the final consistency check).

3.1 Proxy: Client-Facing Proxies and Decryption Oracles

One administrative domain can operate any number of proxies. Each proxy's functionality may be logically divided into two components: handling client requests and, in the case of CR-PDA, serving as decryption oracles for the DB when a particular key should be revealed. None of these proxies need to interact, other than having all client-facing proxies use the same secret s to key the pseudorandom function F and all decryption-oracle proxies use the same public/private key PRX. In fact, these two proxies play different logical roles and could even be operated by two different administrative domains. Currently, all proxies register with a single group membership server, although a fault-tolerant, distributed membership service could be implemented [6].

To discover a client-facing proxy, a client contacts this group membership service, which returns a proxy IP address in round-robin order (this could be replaced by any

technique for server selection, including DNS, HTTP redirection, or a local load balancer). To submit its inputs, a client connects with this proxy and then executes an amortized Oblivious Transfer (OT) protocol on its input batch. This results in the proxy learning $\left\langle E_{\mathrm{DB}}(F_s(k_i)), E_{\mathrm{DB}}(v_i), E_{\mathrm{DB}}(E_{\mathrm{PRX}}(k_i)) \right\rangle$ for each input tuple, with the final element only present for CR-PDA protocols. The proxy pushes this tuple onto an internal queue. (While Section 2.3 only described the use of ElGamal encryption, its special properties are only needed for $E_{\mathrm{DB}}(F_s(k_i))$; the other public-key operations can be RSA, which we use in our implementation.) When this queue reaches a certain length—10,000 in our implementation—the proxy randomly permutes the items in the queue, and sends them to a database server.

Conditional-release PDA protocols have one final step. The database, upon determining that a key k_i's value satisfies f, sends $E_{\mathrm{PRX}}(k_i)$ to a proxy-decryption oracle. The proxy-decryption oracle decrypts $E_{\mathrm{PRX}}(k_i)$ and returns k_i to the database for storage and potentially for subsequent release to other participants in the system.

3.2 Database: Front-end Decryption and Back-end Storage

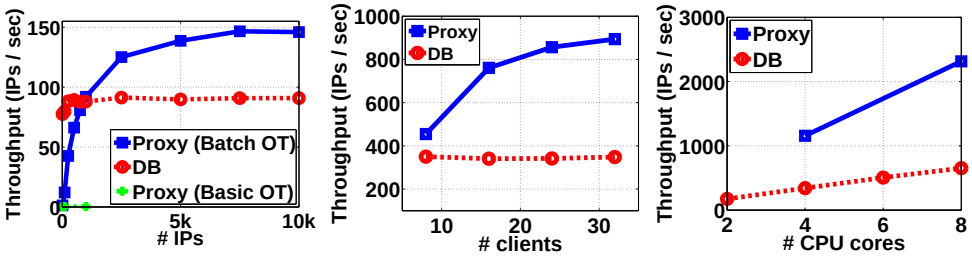
The database component can also be replicated. Similar to the proxy, we separate database functionality into two parts: the *front-end* module that handles proxy submissions and decrypts inputs, and a *back-end* module that acts as a storage layer. Each logical module can be further replicated in a manner similar to the proxy.

The servers comprising the front-end DB tier do not need to interact, other than being configured with the same public/private keypair DB. Thus, any front-end DB can decrypt the $E_{\mathrm{DB}}(F_s(k_i))$ input supplied by a proxy, and the proxies can load balance input batches across these servers.

The back-end DB storage, on the other hand, needs to be more tightly coordinated, as we ultimately need to aggregate all $F_s(k_i)$'s together, no matter which proxy or front-end DB processed them. Thus, the back-end storage tier partitions the keyspace of all 1024-bit strings over all storage nodes (using consistent hashing). All such front-end and back-end DB instances also register with a group membership server, which the front-end servers contact to determine the list of back-end storage nodes. Upon decrypting an input, the front-end node determines which back-end storage node is assigned the resulting key $F_s(k_i)$, and sends the tuple $\left\langle F_s(k_i), v_i, E_{\mathrm{PRX}}(k_i) \right\rangle$ to this storage node (the final element again present only for CR-PDA protocols). As these storage nodes each accumulate a horizontal portion of the entire table T, they can aggregate the values of each table row accordingly. In the case of CR-PDA, they can test the value column for their local table to see if any keys satisfy f. For each such row, the storage node sends the tuple $\left\langle F_s(k_i), T[k_i], E_{\mathrm{PRX}}(k_i) \right\rangle$ to a proxy-decryption oracle.

3.3 Prototype Implementation

Our design is implemented in roughly 5,000 lines of C++. All communication is performed over TCP using BSD sockets, and concurrency is achieved through Linux pthreads. We use the GnuPG library for large numbers (bignums) and cryptographic primitives (*e.g.*, RSA, ElGamal, and AES). The Oblivious Transfer protocol (and its amortized variant) were implemented from scratch, comprising 625 lines of code. All



(a) Scaling # of keys (b) Scaling # of participants (c) Scaling # of proxies/DBs

Fig. 3: Scaling effect of number of (a) keys, (b) participants, and (c) proxy/database replicas.

RSA encryption used a 1024-bit key, and ElGamal used a 1024-bit group size. AES-256 was used in the batch OT and its underlying OT primitive. The back-end DB currently stores table rows only in memory.

4 Performance Evaluation

In this section, we evaluate system *throughput* (number of updates/queries per second) as a function of the number of keys and system participants. We also investigate how throughput scales with greater resources. In each case, we are concerned with both how long it takes for clients to send key-value pairs to the proxy during the OT phase (*proxy throughput*), as well as how long it takes for the DB to decrypt and identify keys with values that satisfy the function f (*DB throughput*). Our experiments were run on multiple machines. The proxy and DB were run on quad-core Intel Xeon 2 GHz machines running CentOS Linux. These machines can perform a 1024-bit ElGamal encryption in 2.2 ms, ElGamal decryption in 2.5 ms, RSA encryption in 0.5 ms, and RSA decryption in 2.8 ms. Clients were run on separate local machines.

As discussed earlier, our system can be used in different contexts. One potential application if collaborative anomaly detection. As modern botnets can range up to roughly 100,000 unique hosts [37], we would like our system to be able to correlate suspicions of hundreds of participating networks within a few hours. Thus, our implementation should be able to process millions of keys in the span of hours, or hundreds of keys per second. We revisit the feasibility of supporting applications in Section 4.2.

4.1 Scaling and Bottleneck Analysis

Effect of number of keys (Figure 3a). Figure 3a measures throughput of a single proxy and DB (each running on a single core) as a function of the number of keys. The throughput of the OT primitive is exceedingly low—less than 1 key per second—and was thus not evaluated on the full range of input sizes. However, when using the amortized OT, proxy throughput significantly improves. Throughput increases with larger numbers of keys per batch, as the amortized OT calls the primitive OT a fixed number of times regardless of the number of input keys. DB throughput, on the other hand, does not increase with larger input batches. The DB must perform a fixed number of decryptions per input tuple—initiated when it receives a batch of encrypted inputs from the proxy—and thus its computational cost is relatively constant per input. Figure 3a shows our DB processes about 90 keys per second (and then becomes CPU limited).

The amortized OT protocol [24] introduces a trade-off between message overhead and memory consumption. The memory footprint of this protocol per client-proxy interaction for n keys is $n \times 32 \times 2 \times 1024/8 = 8196n$ bytes (*i.e.*, we assume 32 bits per key, the 2 values for the OT primitive, and 1024-bit encryption lengths). For $n = 10,000$ keys, for example, this requires 82 MB on both the proxy and the client. To reduce this memory footprint, a user of the protocol could choose to execute the amortized OT protocol in stages by sending k keys at a time.

Effect of number of participants (Figure 3b). We next evaluate the throughput of our system as a function of the number of clients submitting inputs. In this experiment, we limit the proxy and DB to one server machine each. Four client-facing proxy processes are launched on one machine and four front-end DB processes are launched on the other. Figure 3b shows that the proxy scales well with the number of clients, increasing by nearly a factor of two between 8 and 32 clients. When communicating with a single client, a proxy spends a substantial fraction of its time idling (largely while the client is performing its cryptographic operations). The four proxies in this experiment are not CPU limited until they handle 32 clients, at which time the throughput approaches 900 keys per second. The DB, however, is CPU-bound throughout these experiments. It has a throughput of about 350 keys per second, independent of the number of clients (a like amount of work per core as that seen in Figure 3a).

Effect of number of replicas (Figure 3c). Finally, we analyze how our distributed architecture scales with computing resources. Here, we provide up to 8 cores on 2 machines to each of the proxy and DB front-ends. While the proxy functionality alone is evaluated using 64 clients, computing resource constraints meant that the DB (which also required proxies to test) is evaluated using 32 clients. Performance of both the proxy and DB scale linearly with the number of CPU cores allocated to them, enabling a few servers to handle inputs on the order of a few million keys per hour.

Micro-benchmarks. To understand the factors limiting our design's performance, we instrumented the code to account for how CPU cycles are spent. While the DB is entirely CPU bound by the cost of decrypting inputs, the proxy and client engage in the oblivious transfer protocol whose bottlenecks are less clear. When communicating with a single client, we found that the client-facing proxy spends more than 60% of its time idling while waiting for the client (some of the OT time is also spent waiting on clients). The 60% idle time is primarily due to waiting for the client to encrypt k_i and $F_s(k_i)$. The single largest computational expense for the proxy is performing modular exponentiations at 16%; the remaining non-OT tasks add up to 15%. Given that concurrent clients will reduce the proxy's waiting state, achieving higher proxy throughput will require either more efficient cryptographic operations or faster bignum libraries.

We noted earlier that the GnuPG cryptographic library we used performed public-key operations in approximately 2.5–2.8 ms. On the same servers, we benchmarked the Crypto++ library to perform RSA decryption in only 1.2 ms, increasing speed by 130%. Crypto++ would also allow us to take advantage of elliptic curve cryptography (in the situations where homomorphism is not required), which would increase system throughput. In future work, we plan to modify our implementation to use this library.

4.2 Feasibility of Supporting Applications

Anomaly detection. Network operators commonly run systems to detect and localize anomalous behavior by dynamically tracking traffic characteristics. For example, Mao *et al.* [29] found that most DDoS attacks observed within a large ISP were sourced by fewer than 10,000 source IPs, and generated 31,612 alarms over a four-week period (0.8 events per hour). Ramachandran *et al.* [38] found were able to localize 4,963 Bobax-infected host IPs sending spam from a single vantage point. We envision our system could be used to improve the accuracy of these techniques by correlating anomalies across ISP boundaries. This correlation may be done on the level of IP addresses (given DoS attackers typically do not spoof source IPs given ingress filtering [29] and for applications such as email spam that require bidirectional TCP connections), or on the level of subnets. Our system could handle 10,000 IP addresses as keys, with a request rate of several hundred keys per second, even with several hundred participants.

Cross-checking certificates. Multiple vantage points may be used to validate authenticity of information (such as a DNS reply or ssh certificate [34, 41]) in the presence of "man-in-the-middle" attacks. Such environments demand privacy—DNS responses reveal domains that clients access, ssh keys reveal host connection patterns—as well as present scaling challenges due to the potentially large number of keys that could be inserted. Under typical workloads [25, 40] (15 key updates per hour, with 30 keys per participating host), our system scales to support several hundred hosts with a single proxy. Extrapolating out to larger workloads, our system can handle tens of thousands of clients storing tens of thousands of keys with under fifty proxy/database pairs.

Distributed ranking. Search tools such as Alexa and Google Toolbar collect information about user behavior to refine search results returned to users. Users have incentive to install these tools, as they provide benefits (simplified searching and other features). However, they are sometimes labeled as *spyware* as they reveal information about the contents of queries performed by users. Our tool may be used to improve privacy of user submissions to these databases. Alexa Toolbar has an estimated 180,000 active users, and average web users browse 120 pages per day. Roughly extrapolating this data to our results and assuming that users batch their daily usage, our system could handle this daily workload with a single 4-core proxy and DB pair.

5 Conclusions

In this paper, we presented the design, implementation, and evaluation of a collaborative data-analysis system that is both scalable and privacy preserving. Since a fully-distributed solution would be complex and inefficient, our design divides responsibility between a small number of well-known, independent parties—most commonly, a proxy that obliviously blinds the client inputs and a database that aggregates the inputs based on the (blinded) keys. The functionality of both the proxy and the database can be easily distributed for greater scalability and reliability. Experiments with our prototype implementation show that our system performs well under increasing numbers of keys, participants, and proxy/database replicas. The performance is well within the requirements of our motivating applications for collaborative data analysis.

References

1. ALEXA THE WEB INFORMATION COMPANY, 2010. http://www.alexa.com/.
2. ALLMAN, M., BLANTON, E., PAXSON, V., AND SHENKER, S. Fighting coordinated attackers with cross-organizational information sharing. In *HotNets* (Nov. 2006).
3. BEN-DAVID, A., NISAN, N., AND PINKAS, B. FairplayMP: A system for secure multi-party computation. In *CCS* (Oct. 2008).
4. BEN-OR, M., GOLDWASSER, S., AND WIGDERSON, A. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC* (1988).
5. BOGETOFT, P., CHRISTENSEN, D. L., DAMGARD, I., GEISLER, M., JAKOBSEN, T., KRØIGAARD, M., NIELSEN, J. D., NIELSEN, J. B., NIELSEN, K., PAGTER, J., SCHWARTZBACH, M., AND TOFT, T. Secure multiparty computation goes live. In *Financial Crypto* (Feb. 2009).
6. BURROWS, M. The Chubby lock service for loosely-coupled distributed systems. In *OSDI* (Nov. 2006).
7. CHAUM, CREPEAU, AND DAMGARD. Multiparty unconditionally secure protocols. In *CRYPTO: Proceedings of Crypto* (1987).
8. CHOR, B., GOLDREICH, O., KUSHILEVITZ, E., AND SUDAN, M. Private information retrieval. *J. ACM 45*, 6 (Nov. 1998).
9. CRAMER, R., AND DAMGARD, I. On the amortized complexity of zero-knowledge protocols. In *CRYPTO* (Aug. 2009).
10. DACHMAN-SOLED, D., MALKIN, T., RAYKOVA, M., AND YUNG, M. Efficient robust private set intersection. In *ACNS* (2009).
11. DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The second-generation onion router. In *USENIX Technical* (Aug. 2004).
12. DOUCEUR, J. R. The Sybil attack. In *IPTPS* (Mar. 2002).
13. FAGIN, R., NAOR, M., AND WINKLER, P. Comparing information without leaking it. *Comm. ACM 39*, 5 (1996).
14. FRANKLIN, M. K., AND REITER, M. K. Fair exchange with a semi-trusted third party (extended abstract). In *CCS* (Apr. 1997).
15. FREEDMAN, M. J., ISHAI, Y., PINKAS, B., AND REINGOLD, O. Keyword search and oblivious pseudorandom functions. In *TCC* (Feb. 2005).
16. FREEDMAN, M. J., NISSIM, K., AND PINKAS, B. Efficient private matching and set intersection. In *EUROCRYPT* (May 2004).
17. FRIEND-OF-A-FRIEND PROJECT, 2009. http://www.foaf-project.org/.
18. GARRISS, S., KAMINSKY, M., FREEDMAN, M. J., KARP, B., MAZIÈRES, D., AND YU, H. RE: Reliable email. In *NSDI* (May 2006).
19. GOLDREICH, O. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.
20. GOLDREICH, O., MICALI, S., AND WIGDERSON, A. How to play any mental game (extended abstract). In *STOC* (May 1987).
21. GOLDWASSER, S., AND MICALI, S. Probabilistic encryption. *JCSS 28* (1984).
22. GOLDWASSER, S., MICALI, S., AND RACKOFF, C. The knowledge complexity of interactive proof systems. *SIAM J. Computing 18* (1989).
23. HAZAY, C., AND LINDELL, Y. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *TCC* (Mar. 2008).
24. ISHAI, Y., KILIAN, J., NISSIM, K., AND PETRANK, E. Extending oblivious transfers efficiently. In *CRYPTO* (Aug. 2003).
25. JUNG, J., SIT, E., BALAKRISHNAN, H., AND MORRIS, R. DNS performance and the effectiveness of caching. *IEEE/ACM Trans. Networking 10*, 5 (Oct. 2002).

26. KISSNER, L., AND SONG, D. Privacy preserving set operations. In *CRYPTO* (Aug. 2005).
27. LINDELL, Y., AND PINKAS, B. Privacy preserving data mining. In *CRYPTO* (Aug. 2000).
28. MALKHI, D., NISAN, N., PINKAS, B., AND SELLA, Y. Fairplay: A secure two-party computation system. In *USENIX Security* (Aug. 2004).
29. MAO, Z., SEKAR, V., SPATSCHECK, O., VAN DER MERWE, J., AND VASUDEVAN, R. Analyzing large DDoS attacks using multiple data sources. In *SIGCOMM LSAD* (Sep. 2006).
30. NAOR, M., AND PINKAS, B. Oblivious transfer and polynomial evaluation. In *STOC* (May 1999).
31. NAOR, M., AND PINKAS, B. Oblivious transfer with adaptive queries. In *CRYPTO* (Aug. 1999).
32. NAOR, M., AND PINKAS, B. Efficient oblivious transfer protocols. In *SODA* (Jan. 2001).
33. NAOR, M., AND REINGOLD, O. Number-theoretic constructions of efficient pseudorandom functions. In *FOCS* (Oct. 1997).
34. POOLE, L., AND PAI, V. S. ConfiDNS: Leveraging scale and history to improve DNS security. In *WORLDS* (Nov. 2006).
35. PRIVACY RIGHTS CLEARINGHOUSE. A chronology of data breaches, Jan. 2009. `http://www.privacyrights.org/ar/ChronDataBreaches.htm`.
36. RABIN, M. How to exchange secrets by oblivious transfer. Tech. Rep. TR-81, Harvard Aiken Computation Lab. 1981.
37. RAJAB, M. A., ZARFOSS, J., MONROSE, F., AND TERZIS, A. My botnet is bigger than yours (maybe, better than yours): Why size estimates remain challenging. In *HotBots* (Apr. 2007).
38. RAMACHANDRAN, A., AND FEAMSTER, N. Understanding the network-level behavior of spammers. In *SIGCOMM* (Sep. 2006).
39. RINGBERG, H., SOULE, A., AND CAESAR, M. Evaluating the potential of collaborative anomaly detection. Manuscript, 2008.
40. SCHECHTER, S., JUNG, J., STOCKWELL, W., AND MCLAIN, C. Inoculating SSH against address harvesting. In *NDSS* (Feb. 2006).
41. WENDLANDT, D., ANDERSEN, D. G., AND PERRIG, A. Perspectives: Improving SSH-style host authentication with multi-path probing. In *USENIX Technical* (June 2008).
42. YAO, A. C. Protocols for secure computations. In *FOCS* (Nov. 1982).

A Security Assumptions and Definitions

We now motivate and clarify some security assumptions and privacy definitions.

Security against coalitions. We insist on providing security against any coalition of an arbitrary number of participants together with either the database or the proxy. This is essential as otherwise the DB (or proxy) can perform a Sybil attack [12], *i.e.*, create many dummy participants and use their views, together with his own view, to reveal sensitive information. On the other hand, in order to have an efficient and scalable system, we are willing to tolerate vulnerability against a coalition of the DB and the proxy, which could otherwise break participant and keyword privacy.

Power of the adversaries: honest-but-curious vs. malicious adversaries. In our CR-PDA protocol, both proxy and DB are expected to act as HBC. We believe this model is very appropriate for our semi-centralized system architecture. In many deployments, the DB and proxy may be well-known and trusted to act to the best of their abilities, as opposed to simply another participant amongst a set of mutually-distrusted parties.

Of course, these trust assumptions does not extend to the potentially large number of participants, and therefore we require security against any coalition of *malicious participants* (who are allowed to deviate arbitrarily from the protocol). We mention that our PDA protocol provides security even when the DB or proxy are *malicious*. More generally, security holds against any arbitrary coalition of malicious participants that include either a malicious proxy or a malicious DB. Typically, security against fully malicious behavior comes at a great computational cost. We avoid this overhead by providing a weaker notion of security as discussed next.

Notions of security: ideal-real framework vs. input indistinguishability. In cryptography, the security of a protocol is usually defined via the ideal-real framework. Roughly speaking, the protocol should be as secure as an ideal-world implementation in which the players can employ a fully trusted party. This means that any attack that can be carried against the real protocol should be simulatable in the ideal world as well. This notion is very strong, as it shows that the protocol essentially achieves the highest possible level of security.[3] Our CR-PDA protocol provides this notion of security.

A weaker notion (recently studied in [32, 15, 23]) tries to deal separately with privacy and correctness in order to improve efficiency. In particular, malicious parties are allowed to arbitrarily corrupt the correctness of the protocol as long as they do not learn anything about the inputs of honest players. (Formally, this is captured by an indistinguishability-based definition [23].) This is motivated by the fact that a malicious party can often violate semantic correctness in an ideal implementation, *e.g.*, by adding, changing, or omitting inputs to the function—by "lying," in more informal terms. Therefore, it may be reasonable to give up completely on correctness against malicious parties (proxies and DBs) and gain significant computational savings.[4]

Remarks about the PDA and CR-PDA functionalities. The CR-PDA protocol should release those key-value pairs whose values satisfy f. In addition, both protocols release the entire value column ($\mathsf{T}[k_i], \forall i$), with no additional information about the corresponding k_i's. (Alternatively, we support a variant in which only the DB learns the entire value column.) This serves a practical purpose, as it may be hard to fully specify the selection function f *a priori* to collecting clients' inputs. For example, how should an anomaly detection system choose the appropriate frequency threshold τ? In some attacks, 10 observations about a particular IP address may be high (*e.g.*, suspected phishing), while in others, 1000 observations may be necessary (*e.g.*, for bots participating in multiple DoS attacks). Furthermore, a dataset may naturally expose a clear

[3] This does not mean that the protocol always hides all private information, as some information may be leaked by the function itself. For example, consider a PDA computation with only two participants who hold the same input key. The protocol will merge their inputs and output a table with a single row. Thus, each party trivially learns the other's key. This vulnerability follows from the *definition* of $\mathcal{F}$. Hence, this leakage scenario is unavoidable (even in the ideal world) and should not be considered a security violation.

[4] For technical reasons this relaxation makes sense mainly when the malicious parties do not get any output. Since in our PDA functionality only the DB gets an output, we may adopt this relaxed notion and provide *privacy* (at the form of input indistinguishability) against malicious participants and/or malicious proxy, and full *security* (at the form of the ideal-real framework) for coalitions that include a malicious DB.

gap between frequency counts of normal and anomalous behavior; the very reason data operators like to "play" with raw data in the first place.

We also note that the acceptable set of input values and the system's security assumptions has some bearing here. If the domain $\mathcal{D}$ of possible values is large, a client can try to "mark" a key k by submitting it together with an uncommon value $w \in \mathcal{D}$. If a value column that somehow includes w is revealed, the client can discover other clients' values for that same key. That said, a similar problem exists in the PDA functionality—even if the value column is not released to the participants—when one is concerned about collusions between a client and DB (who can search for the $\mathsf{T}[k]$ that includes w). This problem does not arise when the domain is relatively small (*e.g.*, when values are grades over some limited scale). We emphasize that if the domain $\mathcal{D}$ is not small, one can apply the "marking" attack to *any* realization of the functionality, including an *ideal* one (which relies on a fully trusted party), and not only to our implementation.

B The Extended Protocol and Security Proof

Here, we describe the extended protocol of Section 2.2.

1. Each participant interacts with the proxy as follows. For each entry $\langle k_i, v_i \rangle$ in the participant's list, the participant and the proxy run a sub-protocol for encrypted oblivious evaluation of the PRF (EOPRF). At the end of this protocol, the participant learns nothing and the proxy learns only the value $E_{\mathrm{DB}}(F_s(k_i))$. The participant sends the values $E_{\mathrm{DB}}(E_{\mathrm{PRX}}(k_i))$ and $E_{\mathrm{DB}}(v_i)$ together with a proof of knowledge (POK) for knowing the plaintext of the last entry. If the POK succeeds, then the proxy re-randomizes the ciphertexts and adds the triple to a list. Otherwise, if the POK fails, the proxy ignores the triple.

2 and 3. Same as in the original protocol.

4. The DB builds the tables R and H as in the original protocol. For each row in R, the DB sends to the proxy the value $F_s(k_i)$ together with the corresponding list $\mathsf{E}[k_i]$ which supposedly contains ciphertexts of the form $E_{\mathrm{PRX}}(k_i)$. The DB also re-randomizes these ciphertexts.

5. The proxy goes over R and, for each entry, it decrypts all the values in the list $\mathsf{E}[k_i]$ and verifies that the plaintext corresponds to the blinded key $F_s(k_i)$. It reports inconsistencies to the DB and sends k_i if it appears in the list $\mathsf{E}[k_i]$.

6. For each row, the DB updates the list $\mathsf{T}[k_i]$ by omitting the values v_i for which inconsistencies were found. Then, it applies f again to the updated row, checks whether it should be released, and, if so, publishes the corresponding key k_i together with the updated list of values $\mathsf{T}[v_i]$. (The value k_i was given by the proxy as at least one of the ciphertexts in $\mathsf{E}[k_i]$ was consistent with the blinded key.)

We now sketch the proofs for the security of the protocol. First let us formally define the functionality we consider. Consider all submitted key-value pairs as a table, where each distinct key k_i is associated with a list $\hat{\mathsf{T}}[k_i]$ of all values v_i submitted with it. Let $\hat{\mathsf{R}}$ be the sub-table that consists of all the rows that should be revealed (according to f), and let $\hat{\mathsf{H}}$ be the table that contains all the other entries with the key column omitted. Our functionality outputs $\hat{\mathsf{R}}$ as a public value and $\hat{\mathsf{H}}$ as a private output for the DB. We prove that our protocol securely computes this functionality.

Honest-but-curious coalition of participants and a proxy. The joint view of the proxy and the HBC participants contains the following: (1) the inputs (k_i, v_i) of the HBC participants and the public outputs $\hat{\mathsf{R}}$; (2) the information exchanged by the proxy and the HBC participants during the first stage; (3) the view of the proxy when interacting with other participants in the first stage, which consists of the proxy's view of the sub-protocols (EOPRF and POK) as well as triples of the ciphertexts $E_{\text{DB}}(v_i)$, $E_{db}(F_s(k_i))$, and $E_{\text{DB}}(E_{\text{PRX}}(k_i))$; and (4) the table R sent by the DB to the proxy at the "revealing" phase of the protocol.

This view can be simulated, given the corresponding inputs (k_i, v_i) and the outputs $\hat{\mathsf{R}}$, as follows. Choose a random PRF key s, as well as public keys PRX and DB. Simulate (1) and (2) in the natural way (all the information needed for these computations is given). To simulate (3), use the simulators of the sub-protocols and generate garbage ciphertexts $E_{\text{DB}}(0), E_{\text{DB}}(0), E_{\text{DB}}(0)$. To simulate (4), encrypt the values in $\hat{\mathsf{R}}$ under PRX and blind the keys under s.

Honest-but-curious coalition of participants and a DB. The joint view of the proxy and the HBC participants contains the following: (1) the inputs (k_i, v_i) of the HBC participants and the public outputs $\hat{\mathsf{R}}$; (2) the view of the HBC participants during the interaction with the proxy, which consists of the view of the sub-protocols (EOPRF and POK) as well as triples of ciphertexts $E_{\text{DB}}(v_i)$, $E_{db}(F_s(k_i))$, and $E_{\text{DB}}(E_{\text{PRX}}(k_i))$; and (3) the view of the DB when interacting with the proxy, which consists of the tables R and H (encrypted under the DB's public key).

Given the corresponding inputs (k_i, v_i), the public output $\hat{\mathsf{R}}$, and the DB's private output $\hat{\mathsf{H}}$, we show how to simulate the above view. First, choose a random PRF key s, as well as public keys PRX and DB. Then, simulate (1) and (2) in the natural way (all the information needed for these computations is now given). It remains just to simulate R and H. The table R can be computed from $\hat{\mathsf{R}}$ and s. To simulate H, we should somehow add blinded values to $\hat{\mathsf{H}}$ (and encrypt the tuples under DB). We do this by building a key-value table for the inputs of the HBC participants. Then, for each row k_i, we choose a random consistent row in $\hat{\mathsf{H}}$ and add the value $F_s(k_i)$ as an additional blinded-key column. (A row is consistent with a key k_i if the list of values of the HBC's that are associated with k_i appear as part of the value list of the row in $\hat{\mathsf{H}}$.) Finally, for those rows which are left with no blinded key column, a random value is added.

Malicious coalition of participants. Let A be an adversarial strategy for a coalition of cheating participants. We construct a simulator that achieves the same "cheating" affect in the ideal world. The simulator S chooses a key s for the PRF, as well as pairs of private/public keys for the DB and proxy. It provides these keys to A and executes A. For each iteration i, A generates a triple (a_i, b_i, c_i), together with a POK for knowing the plaintext encrypted in c_i. (In an honest execution $a_i = E_{\text{DB}}(F_s(k_i)), b_i = E_{\text{DB}}(E_{\text{PRX}}(k_i))$, and $c_i = E_{\text{DB}}(v_i)$, for some k_i and v_i.) The simulator S uses the POK to extract v_i; if the POK fails, then S ignores the triple. Finally, S checks (using all the above keys) whether a_i and b_i are consistent (*i.e.*, it decrypts a_i to a'_i, decrypts b_i to b'_i, and then verifies that $F_s(b'_i) = a_i$). If the check fails, S ignores the tuple. Otherwise, S, which now knows both k_i and v_i, passes these entries to the trusted party.