The Dissertation Committee for Trinabh Gupta
certifies that this is the approved version of the following dissertation:

Toward practical and private online services

Committee:

Lorenzo Alvisi, Supervisor

Simon Peter

Michael Walfish

Emmett Witchel

Toward practical and private online services

by

Trinabh Gupta

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

August 2017

Acknowledgments

It gives me immense pleasure to finally thank the people who helped me write this dissertation, and who spurred me during grad school.

First, I would like to thank Mike Walfish, who advised me from day one of grad school. Just like all great advisors, Mike helped me develop my research skills: how to precisely formulate a problem and nail down its solution, what kind of research problems to pursue, how to reduce bias in experimentation, etc. However, what has stood out is Mike's immense attention to detail—the little things, and the energy he has put into training me. He very patiently coached me on my oral and written communication skills. I can never forget the time we spent writing and rewriting "bullets" for Popcorn, a system described in this dissertation. Looking back, I think this writing exercise made me mentally stronger, helped me raise the quality of my work, and propelled me toward being an independent researcher. Mike also went to great depths to assist me in my job hunt, and I cannot thank him enough for that. I believe that graduate school would not have been as much fun (and pain ☺), I would not have been the person I am today, and my research would not be as careful as it has been, had it not been for Mike's efforts. Thank you, Mike, once again, for everything! You deserve a lot of credit.

Another person who greatly helped and inspired me is Lorenzo Alvisi. I took Lorenzo's distributed computing class during my first semester. This class is one of my all-time favorite. Not only did I learn a whole lot about distributed systems and algorithms, but Lorenzo's passion and energy rubbed off on me. I was very fortunate that Lorenzo agreed to co-advise me when Mike moved to NYU. Lorenzo continued to help me improve my research skills. In particular, he instilled the importance of zooming out and looking at the bigger picture. His perspective is the best I have come across. I am grateful to Lorenzo for his help during the "dark days" of my PhD. Lorenzo, you might not realize this, but your optimism and stories greatly helped me tick along during periods when there was no visible research progress. Thank you, Lorenzo!

Next, I want to thank the students who helped with the research presented in this dissertation. Natacha Crooks helped design, implement, and draft Popcorn (Chapter 2). Whitney Mulhern wrote a client-side library for Popcorn. This library helped us demonstrate that Popcorn's prototype is reasonably mature. Srinath Setty has been my go-to person whenever I have needed to freshen up my thinking. He also helped me significantly improve the evaluation section of Popcorn. Finally, Henrique Fingler's contributions to the implementation of Pretzel (Chapter 3) were crucial for Pretzel's successful submission to SIGCOMM.

I would like to thank my thesis committee members for helping me improve this dissertation. Simon Peter and Emmett Witchel gave me valuable feedback on my defense talk, and on various drafts of this thesis. The questions they asked during my talk motivated Chapter 4 of this thesis.

Several other people provided extensive comments on the systems that this thesis describes. I would like to thank Carlos Aguilar-Melchor, Eric Crockett, Bryan Ford, Yuval Ishai, James Mickens, Thomas Schneider, Vitaly Shmatikov, Emmett Witchel, and Samee Zahur for their help and comments at various points in the development of Popcorn and Pretzel. I would also like to thank the anonymous reviewers of SOSP-2015, NSDI-2016, NSDI-2017, and SIGCOMM-2017 for their valuable reviews.

For the first two years of grad school, I had the comfort of having a senior student like Joshua Leners besides me. Josh helped me understand the intricacies of systems research. He taught me how to write an "evaluation plan" and how to rigorously evaluate a research prototype. These lessons turned out to be very useful when I was evaluating Popcorn and Pretzel.

Amar Phanishayee, Ratul Mahajan, and Jaeyeon Jung mentored me when I interned at Microsoft Research (Summer 2013). They introduced me to research at the intersection of systems and security. I highly enjoyed my time at Microsoft—the interactions with my mentors, the discussions with other members of the systems research group, and the intern activities. After my internship, Amar continued to mentor me informally. He also was a great help and inspiration during the job application and interview process. Thank you, Amar, for sharing your enthusiasm, and for all the valuable interview tips!

While in graduate school I had the joy of interacting with sharp and talented peers, at both UT Austin and NYU. At UT Austin, I would like to thank Sebastian Angel, Apurv Bhartia, Alan Dunn, Manos Kapritsos, Venkata Koppula, Sangman Kim, Youngjin Kwon, Michael Lee, Syed Akbar Mehdi, Ashay Rane, Chunzhi Su, Yang Wang, Chao Xie, Yuanzhong Xu, and Sangki Yun for their encouragement and support. When I visited NYU, as a research fellow, Talal Ahmad, Varun Chandrasekaran, Shiva Iyer, Cheng Tan, Ashwin Venkataraman, Riad Wahby, and Zhaoguo Wang helped me bear the stress of the transition.

I would also like to thank the administrators who helped maintain my paperwork: Lindy Aleshire, Leslie Cerve, Katie Dahm, Lydia Griffith, and Leah Wimberly. Lindy, in particular, contributed to procure AWS resources to run Popcorn's experiments, and she helped put together a request to obtain datasets for Pretzel's experimental evaluation. Funds from Mike's and Lorenzo's research grants, and a graduate dissertation fellowship from UT Austin supported me financially as a PhD student.

A PhD is hard to navigate without a strong support system. I would like to thank my parents, Anuj and Archana, for encouraging me to pursue a PhD, and firmly backing me through thick and thin. I also thank my sister, Ekawali, for her love and affection. My extended family in the United States and my parents-in-law, Sudhir and Neena, were also a source of encouragement. Sudhir helped me get out of tight spots, by sharing perspective and experience from his days of being a PhD student, a professor, and a dean. Finally, a special thank you to my wife, Kanika, who supported me unconditionally throughout my PhD, often putting my career in front of hers.

TRINABH GUPTA

The University of Texas at Austin
August 2017

Toward practical and private online services

Publication No. ____________________

Trinabh Gupta, Ph.D.
The University of Texas at Austin, 2017

Supervisor: Lorenzo Alvisi

Today's common online services (social networks, media streaming, messaging, email, etc.) bring convenience. However, these services are susceptible to privacy leaks. Certainly, email snooping by rogue employees, email server hacks, and accidental disclosures of user ratings for movies are some sources of private information leakage. This dissertation investigates the following question: Can we build systems that (a) provide strong privacy guarantees to the users, (b) are consistent with existing commercial and policy regimes, and (c) are affordable?

Satisfying all three requirements simultaneously is challenging, as providing strong privacy guarantees usually necessitates either sacrificing functionality, incurring high resource costs, or both. Indeed, there are powerful cryptographic protocols—private information retrieval (PIR), and secure two-party computation (2PC)—that provide strong guarantees but are orders of magnitude more expensive than their non-private counterparts. This dissertation takes these protocols as a starting point and then substantially reduces their costs by tailoring them using application-specific properties. It presents two systems, *Popcorn* and *Pretzel*, built on this design ethos.

Popcorn is a Netflix-like media delivery system, that provably hides, even from the content distributor (for example, Netflix), which movie a user is watching. Popcorn tailors PIR protocols to the media domain. It amortizes the server-side overhead of PIR by batching requests from the large number of concurrent users retrieving content at any given time; and, it forms large batches without introducing playback delays by leveraging the properties of media streaming. Popcorn is consistent with the prevailing commercial regime (copyrights, etc.), and its per-request dollar cost is 3.87× that of a non-private system.

The other system described in this dissertation, Pretzel, is an email system that encrypts emails end-to-end between senders and intended recipients, but allows the email service provider to perform content-based spam filtering and targeted advertising. Pretzel refines a 2PC protocol. It reduces the resource consumption of the protocol by replacing the underlying encryption scheme with a more efficient one, applying a packing technique to conserve invocations of the encryption algorithm, and pruning the inputs to the protocol. Pretzel's costs, versus a legacy non-private implementation, are estimated to be up to 5.4× for the email provider, with additional but modest client-side requirements.

Popcorn and Pretzel have fundamental connections. For instance, the cryptographic protocols in both systems securely compute vector-matrix products. However, we observe that differences in the vector and matrix dimensions lead to different system designs.

Ultimately, both systems represent a potentially appealing compromise: sacrifice some functionality to build in strong privacy properties at affordable costs.

Contents

Previously published material

This dissertation describes two systems which have appeared elsewhere.

We describe Popcorn in Chapter 2 by revising [130]:

- T. Gupta, N. Crooks, W. Mulhern, S. Setty, L. Alvisi, and M. Walfish. Scalable and private media consumption with Popcorn, In USENIX NSDI, Mar. 2016.

And we describe Pretzel in Chapter 3 by revising [131]:

- T. Gupta, H. Fingler, L. Alvisi, and M. Walfish. Pretzel: Email encryption and provider-supplied functions are compatible, In ACM SIGCOMM, Aug. 2017.

The author of this dissertation conceived, initiated, designed, implemented, evaluated, and led work on both the described systems.

Chapter 1

Introduction

Privacy is essential. But today's common online services undermine it, while focusing on providing conveniences to users. A relevant question is, how can one use these services while hiding the content of one's requests, from both the network and the service provider?

The motivation for this question is a fundamental tension in the ecosystem in which online services operate. On one side are users deeply uncomfortable with exposing the content of their requests (for example, which movie they are requesting, the body of the email they are sending, etc.), in particular to centralized servers. The discomfort is partly philosophical: they argue that freedom requires the ability to use online services privately [212]. But the discomfort is also practical. An entity with access to, say, a person's media consumption profile, can reveal the person's sexual orientation, political leaning, cultural affiliations, etc. [187, 188, 219]. Moreover, centralized services are subject to a wide range of insider and outsider threats. Reputable organizations have been known to unwittingly harbor rogue employees bent on gaining access to user email accounts and other private user information [50, 200, 262]. And well-run organizations are not immune from hacks [245, 246]—nor, indeed, from the law. Just in the first half of 2013, Google [121], Microsoft [185] and Yahoo! [250] collectively received over 29,000 requests for email data from law enforcement, and in the overwhelming majority of cases responded with some customer data [184].

On the other side are online service providers, who provide their services within a commercial framework, and need to be compensated. These providers run *functions* on user data. Some of these functions target revenue generation (for example, topic extraction [125] and advertisements), while others target the quality of user experience (for example, spam filtering, email search, predictive personal assistance [52, 73, 96, 198], low-delay media playback, and movie recommendations). But independent of their exact purpose, functions rely on the content of user requests.

The general tension between privacy and profit has been noted in many quarters before. Government regulators, including from the US, Canada, and the EU, are on the record in support of an idea known as *Privacy by Design* (PbD) [17, 19, 42, 43, 211]. PbD is the abstract philosophy that software products should build in privacy up front rather than treating it as an afterthought. However, the apparent consensus among regulators notwithstanding, no one seems to know what PbD truly means, much less how to implement it [132, 207].

In principle, it is possible to implement online services in a way that the content of user requests is hidden, and yet providers are able to run functions on that content. One option is to use cryptographic protocols. Consider an email service. An email sender can encrypt an email using an encryption scheme, such as PGP [266], and the intended recipients can decrypt and obtain email contents. Then, the email provider and each recipient can engage in a *secure two-party computation* (2PC); the term refers to cryptographic protocols that enable one or both parties to learn the output of an agreed-upon function (spam filtering, topic extraction, etc.), without revealing the inputs (proprietary spam and topic models, email contents, etc.) to each other.

So why don't such privacy-preserving services exist? After all, software that implement cryptographic primitives have long existed: PGP [266] was first implemented in 1991; implementations of secure two-party computation have existed at least since 2004 [176]. There are several reasons: general deployment difficulties (it's hard, for example, to modify a communication medium as entrenched as email [109]), usability (cryptographic primitives require keys; how should users share keys across devices and find each other's keys?), and even politics (there are entrenched interests who would prefer widespread surveillance and data collection). However, one primarily technical reason is high resource costs: despite continuous progress, the state-of-the-art implementations of cryptographic primitives, if applied as they are described in the literature, result in systems whose resource consumption (CPU time, disk I/O, and network transfers) is orders of magnitude higher than in systems without the primitives. And the reasons for these high costs are fundamental. For example, if a user wants to hide which movie she is watching, then her request for the movie must prompt the service provider to touch its entire video library (because otherwise the provider can infer which movie the user is *not* interested in).

Indeed, privacy, functions, and costs are in a three-way trade-off. This dissertation does not attempt to eliminate this trade-off—it can't. Rather, it attempts to improve the trade-off.

1.1 Problem statement and approach

This dissertation studies how two online services that are in common use today—Netflix-like media delivery and email—can gain significantly on privacy while limiting sacrifices on functions and costs. More specifically, it asks, how can we build systems (Netflix-like media delivery and email) that

(i) *provide strong privacy*, that is, hide the content of user requests from both a network eavesdropper [24, 107] and the service provider,

(ii) *support basic functions consistent with existing commercial arrangements*, that is, make only limited sacrifices in functionality, and implement functions in a way that they remain consistent with existing commercial and policy regime, and

(iii) *have low cost*, that is, have overheads (resource consumption, or this consumptions converted to a dollar amount) that are small multiples of the costs in the status quo?

To answer this question, we start with cryptographic primitives (they meet the first two requirements), and then bring practicality, that is, reduce costs, by following the design ethos of tailoring cryptographic protocols to the application.

1.2 Contributions and contents

This dissertation focuses mainly on describing two systems—Popcorn for Netflix-like media delivery and Pretzel for email—that build on the design ethos described above.

Popcorn (Chapter 2). Popcorn is the first (to our knowledge) Netflix-like media delivery system that provably hides, even from the content distributor (e.g., Netflix), which movie a user is watching; respects copyrights, and achieves plausibly deployable performance.

Popcorn starts from private information retrieval (PIR) cryptographic protocols; these protocols [71, 161] allow a client to retrieve an object from a server's library without revealing to the server which object is retrieved. Using novel techniques, Popcorn addresses several challenges of applying PIR to media delivery. First, it composes two different types of PIR protocols to address a tension between content protection and server-side CPU overhead; in contrast, prior work that composes the two types of PIR focuses on minimizing communication costs. Second, it amortizes the high server-side disk I/O overhead over multiple requests from different users by splitting the library into exponentially-increasing slices (each slice contains a piece from every movie), configuring the first slice to be "narrow" to keep playback delay small, applying PIR independently to each slice, and processing requests to the slices in batches. Prior PIR implementations can batch client requests but not for applications that are delay-sensitive. Third, it combines padding, splitting, and domain-specific compression techniques to convert variable-sized media objects to the same size.

Popcorn's evaluation provides a rough estimate of how much a private Netflix-like service might cost in terms of computational resources. For each request, Popcorn consumes 1080× more CPU time than a non-private baseline, about 14× more I/O bandwidth, and 2× longer network transfers. However, since CPU is cheap, these overheads, when translated to dollars, are 3.87× that of the non-private baseline.

Pretzel (Chapter 3). Pretzel[1] is the first (to our knowledge) system that encrypts emails between senders and recipients, and simultaneously allows the email provider to perform spam filtering and topic extraction over them, using linear classifiers from machine learning (Naive Bayes, Support vector machines, logistic regression), at tolerable cost.

Pretzel starts with a relatively efficient 2PC protocol geared to computations that consist mostly of linear operations. It addresses two issues with a naive instan-

[1] Pretzel (Chapter 3) can be read independently of Popcorn (Chapter 2).

tiation of this protocol: high resource consumption and leakage of private data in the case that one or both parties deviate from the protocol. To reduce resource consumption, it replaces the underlying encryption scheme with a more modern and efficient one, conserves invocations of the encryption algorithm by applying a packing technique, and decomposes classification into a public and a private step. The third refinement requires a novel 2PC (sub)protocol to compute a vector-submatrix product, where one party's private input is the (full) matrix, and the other party's private inputs are the vector and a set indicating the submatrix of the full matrix.

We evaluate Pretzel for realistic parameter values in the email setting [123]. Pretzel's CPU and network overheads, versus a legacy non-private implementation, are up to $5.4\times$ for the provider. However, Pretzel requires additional client-side resources of several hundred megabytes of storage and per-email CPU time of several hundred milliseconds. Pretzel's overheads are certainly substantial, but within the realm of plausible deployability.

Comparisons and connections (Chapter 4). This chapter describes a framework to situate and compare the various approaches to answering our question in Section 1.1. In particular, we look at approaches that build on trusted hardware (for example, Intel SGX) and fully homomorphic encryption (FHE), as embodying very different trade-offs. Chapter 4 also discusses fundamental connections between Popcorn and Pretzel. Both these systems securely compute a vector-matrix product, using additively homomorphic encryption. However, differences in the dimensions of the vectors and matrices in the two systems result in significantly different designs.

Summary (Chapter 5). Chapter 5 summarizes and concludes this dissertation, and discusses the lessons we learned while building Popcorn and Pretzel.

Chapter 2

Popcorn: A private media delivery system

Media delivery systems differ widely. YouTube's library, for instance, is large, continuously updated, freely distributed, and supported by advertising. Netflix's library is comparatively small, updated infrequently [15], subject to strict content protection, and supported by paid subscriptions. This chapter describes Popcorn, a Netflix-like system, that adopts the following privacy, functionality, and cost requirements:

- It must comprehensively and provably hide the content of client requests. "Comprehensive" means hiding consumption not only from a network eavesdropper [24, 107] but also from the content distributor. "Provable" means that we wish to avoid the risk [49] of heuristic solutions.
- It must disseminate content in an on-demand fashion (that is, support low-delay playback), and respect current controls on content dissemination (copyrights, etc.).
- It must dispense privacy at an attractive price point. The resource cost converted into dollars should be within a small multiple of what customers pay to access content today.

At first blush, Tor [94] and other anonymity systems [5, 165] (which conceal *who* consumes content) satisfy the above requirements. However, these solutions conflict with commercial media delivery: now Netflix would have to rely on the altruism of Tor nodes. Moreover, the capacity, latency, and reliability on a Tor network is unlikely to match the requirements of Netflix.

Thus, Popcorn turns to a large body of cryptographic protocols known as *Private Information Retrieval*, or *PIR* (§2.1.2). These protocols [72, 110, 161, 192, 253] allow clients (content consumers) to request content from servers (content distributors) without the servers being able to infer which items the clients requested.

PIR protocols are promising as they meet the first requirement. However, these protocols do not satisfy the other two requirements due to several challenges (linear overhead of PIR, requirement of fixed object sizes, etc. as described in Section 2.2). Popcorn addresses these challenges by cherry-picking techniques (batching, etc.) from the literature on PIR (§2.6) and working through the systems ramifications of tailoring them to the specific domain of Netflix-like media consumption (§2.3). But before we can delve into the details, we must give some necessary background on PIR protocols, and describe the setting and assumptions underlying Popcorn.

2.1 Setting and background on PIR

2.1.1 Scenario and threat model

The media delivery ecosystem has three principals: a *content creator*, a *content distributor*, and a *content consumer*. The creator (e.g., a movie studio), delegates to the distributor (e.g., an online streaming service like Netflix) the tasks of disseminating content and charging consumers.

We model the content kept by the distributor as a collection L of n objects; we call L the *library*. We assume that a mapping, between the integers $1, \ldots, n$ and the names of the objects in L, is known to the distributor and the consumers. Therefore, a consumer can select a specific object by supplying the corresponding integer.

Threat model. We consider an attacker (for example, the content distributor or a network eavesdropper) trying to infer what object the consumer is accessing. The attacker has full access to the network and to the content of the consumers' requests, but for two restrictions. First, we do not consider side-channel attacks that, for example, use knowledge of where individual consumers pause playback, or of their concurrent web browsing activity. Second, we assume the existence of two non-colluding servers that the distributor can use to serve content. To satisfy this assumption in practice, one can pick servers from separate administrative domains (e.g., from different CDNs [32]). We discuss this topic further in Section 2.7.

We assume, as do today's media delivery systems [38, 95], that the client-side media decode and display environment can defeat content consumers intent on copying and redistributing content beyond what the distributor allows.

Finally, we treat content integrity as an orthogonal problem that undermines correctness (§2.1.2) but not privacy. The literature offers standard solutions to guarantee content integrity (content hashing, etc.).

2.1.2 Private Information Retrieval (PIR)

The high-level goal of PIR protocols aligns with that of Popcorn: they allow a client to use an integer between 1 and n to retrieve any object from a library L of n ℓ-bit objects kept by a set of k servers ($k \geq 1$) without leaking to the servers any information about which object was retrieved. A PIR protocol is structured around three procedures: Query, Answer, and Decode. To privately retrieve object $O_b = L[b]$ ($1 \leq b \leq n$), the client invokes $\mathsf{Query}(b)$ to produce k query vectors $q_1, \ldots, q_k$, one for each server, and forwards q_j to server S_j ($1 \leq j \leq k$). Each S_j replies with a_j = $\mathsf{Answer}(q_j, L)$. Finally, the client computes $O_b = \mathsf{Decode}(a_1, \ldots, a_k)$ by applying the decode algorithm to the servers' responses.

We want three properties from a PIR protocol:

- ***Correctness.*** If a client requests the object in library L with index b, then the protocol indeed provides it with object $L[b]$.
- ***Privacy.*** After the server sees a query vector, its probability of guessing the client's requested index is no better than if the server had not seen the query in the first place. This property can be generalized to coalitions of $t < k$ servers, requiring that any t out of k servers jointly do not learn any information about the index of the requested object.
- ***Communication efficiency.*** The size of a server's reply must not be much larger than ℓ, and the size of a client's request must be far smaller than ℓ (though acceptable if there is some overhead above the minimum query size of $\log_2 n$ bits).

We discuss below two such PIR protocols.

```
Query (index $b$):
    for $i = 1$ to $n$ do
        $f \leftarrow (i == b)\ ?\ 1 : 0$
        $c_i \leftarrow \mathsf{Enc}(pk, f)$
    return $q = (pk, c_1, \ldots, c_n)$
Answer (query vector $q$, library $L$):
    // Represent $L$ as a matrix of $y$-bit integers:
    // $L \in (\{0,1\}^y)^{n \times (\ell/y)}$
    for $j = 1$ to $\ell/y$ do
        $r_j \leftarrow \prod_{i=1}^{n} {c_i}^{L_{i,j}}$
    return $a = (r_1, \ldots, r_{\ell/y})$
Decode (answer $a$, secret key $sk$):
    return $\mathsf{Dec}(sk, r_1), \ldots, \mathsf{Dec}(sk, r_{\ell/y})$
```

Figure 2.1: A computational PIR (CPIR) protocol based on an additively homomorphic cryptosystem (Gen, Enc, Dec) and due to Stern [226]. (pk, sk) is a (public, private) key pair generated using Gen. n is the number of objects in the library L, and ℓ is the length of each object.

2.1.3 Computational PIR (CPIR) protocols

CPIR protocols [161] require only a single, computationally bound server ($k = 1$). They are commonly constructed using additively (and not fully [58, 111]) homomorphic public key cryptosystems. A cryptosystem is *additively homomorphic* if $\mathsf{Dec}(sk, \mathsf{Enc}(pk, m_1) \cdot \mathsf{Enc}(pk, m_2)) = m_1 + m_2$, where m_1, m_2 are plaintext messages, $+$ represents addition of two plaintext messages, $\cdot$ is a binary operation (for example, addition, multiplication, etc.) on the ciphertexts, (pk, sk) is a (public, private) key pair generated using the key generation algorithm Gen, Dec is the decryption algorithm, and Enc is the encryption algorithm. Note that Enc is randomized; thus, repeatedly encrypting the same plaintext produces different ciphertexts. Examples of cryptosystems used in CPIR are the Paillier [193] and the lattice-based Ring-LWE [59].

Figure 2.1 depicts a CPIR protocol, due to Stern [226], that meets the three properties (§2.1.2):

- ***Correctness.*** $\mathsf{Dec}(sk, r_j) = \mathsf{Dec}(sk, \prod_{i=1}^{n} {c_i}^{L_{i,j}})$, which equals $\sum_{i=1}^{n} \mathsf{Dec}(sk, c_i) \cdot L_{i,j}$ after the application of the additively homomorphic property. But $\forall i \in \{1, \ldots, n\} \setminus$

```
Query (index $b$):
    // Generate the first $k-1$ query vectors randomly
    for $j = 1$ to $k-1$ do
        select $q_j \in_R \{0,1\}^n$
    $e_b \leftarrow$ an $n$-bit string with all zeros except at $b$-th position
    $q_k \leftarrow e_b \oplus q_1 \oplus \cdots \oplus q_{k-1}$                // $\oplus$ is bit-wise XOR
    return $q_1, \ldots, q_k$
Answer (query vector $q$, library $L$):
    // $q$ is one of the outputs of Query
    // $L$ has $n$ objects; each is $\ell$ bits
    // $q$ is a row vector, $L$ a logical matrix: $L \in \{0,1\}^{n \times \ell}$
    return $q \cdot L$                // product over the two-element field $\mathbb{F}_2$
Decode (answers $a_1, \ldots, a_k$):
    // $a_j$ is the output of Answer
    return $a_1 \oplus \cdots \oplus a_k$
```

Figure 2.2: The ITPIR protocol of CGKS [72]. n is the number of objects in library L, and ℓ is the length of each object. k is the total number of servers. (In Popcorn, k=2.)

b, $\mathsf{Dec}(sk, c_i) = 0$, by construction of c_i. Similarly, $\mathsf{Dec}(sk, c_b) = 1$. Therefore, $\mathsf{Dec}(sk, r_j) = \mathsf{Dec}(sk, c_b) \cdot L_{b,j} = L_{b,j}$.

- ***Privacy.*** The guarantee that server S does not learn b hinges on S being computationally bounded. All S sees is $q = (pk, c_1, \ldots, c_n)$. If S could systematically guess b (that is, guess which ciphertext is $c_b = \mathsf{Enc}(pk, 1)$), then S could likewise guess which entry is the encryption of 1 (versus 0)—which would contradict the properties of the underlying encryption scheme.
- ***Communication efficiency.*** The length of the server's reply is $(\ell/y) \cdot |c|$ bits, where ℓ/y is the number of ciphertexts in the reply and $|c|$ is the size (in bits) of a ciphertext. $(\ell/y) \cdot |c|$ is comparable to ℓ, the size of object O_b, if the expansion ratio, $|c|/y$, of the underlying additively homomorphic cryptosystem is small.[1] The client's request contains n ciphertexts and is thus $|c| \cdot n$ bits. When $\ell \gg n$ (as will be the case in our context) and $|c|$ is a small constant (e.g., 2048 in many Paillier implementations), $|c| \cdot n$ is much smaller than ℓ.

[1] The Paillier cryptosystem has a message expansion ratio of ≥ 2.

2.1.4 Information-theoretic PIR (ITPIR) protocols

ITPIR protocols [72] use more than one server ($k > 1$), and assume that they do not collude; thus, in practice, the servers must belong to different administrative domains.

Figure 2.2 shows the CGKS [72] ITPIR protocol. It meets the three properties of PIR (§2.1.2):

- ***Correctness.*** The output of Decode is $\bigoplus_{j=1}^{k} a_j$, which equals $\bigoplus_{j=1}^{k}(q_j \cdot L)$. By properties of the field $\mathbb{F}_2$ (that addition is XOR and that multiplication distributes over addition), $\bigoplus_{j=1}^{k}(q_j \cdot L) = (\bigoplus_{j=1}^{k} q_j) \cdot L = e_b \cdot L = L[b]$.
- ***Privacy.*** Each server in $S_1, \ldots, S_{k-1}$ sees a randomly generated query vector, and therefore each server (and all of them combined) cannot learn any information about b. Server S_k sees q_k, which is constructed by XORing unit vector e_b with the one-time pad $q_1 \oplus \cdots \oplus q_{k-1}$. By the properties of one-time pads, S_k can learn information about e_b only by learning the one-time pad (or by colluding with all other servers).
- ***Communication efficiency.*** The combined length of the servers' reply is $k \cdot \ell$ bits. In Popcorn, we set $k = 2$ to keep this comparable to ℓ, the size of an object. A client's request consists of k n-bit-long query vectors, which is much smaller than ℓ when k is small.

2.2 Challenges of applying PIR

Though PIR is promising, there are a number of challenges in applying it to large-scale media consumption:

- *Resources.* The I/O and CPU resources required to serve a single request are proportional to the size of the library. Batching requests should help amortize some of this overhead, but it is in tension with the next issue.
- *Strict deadlines.* Media delivery has stringent latency requirements: initial delay must be small, and the delivery must obey real-time constraints.
- *Variable object sizes.* Object sizes vary as a function of encoding or playback time. However, PIR assumes objects of identical size.
- *Content protection in ITPIR vs. CPIR.* Content creators may be loath to disseminate the content beyond its original distribution channel. Yet ITPIR requires multiple

	I/O	CPU	content prot. (ITPIR)	resists collusion	object sizes	pricing, reco
XPIR [34]		◐	●	●		
RAID-PIR [86]					◐	
Percy++ [116]		◐	◐	◐	◐	●
Popcorn	●	◐	●		◐	

Figure 2.3: Prior PIR-oriented works (rows) and which media-related challenges they address (columns), assuming two servers for ITPIR-based works. ● means that the work addresses the challenge; ◐ means that it partially addresses the challenge.

non-colluding servers, and hence multiple administrative domains, necessitating such dissemination. CPIR, on the other hand, requires only a single server; however, its computational cost is significantly higher.[2]

- *Billing, access control, and recommendations.* For business reasons, media services may need to support access control, pricing policies (tiers, etc.), targeted advertising, and recommendations. Yet, private retrieval conflicts with all of this functionality.

Subsets of these challenges have been addressed before (Figure 2.3). Popcorn aims mainly at the resource consumption issue, via the architecture and design described next.

2.3 Architecture and design of Popcorn

Figure 2.4 depicts Popcorn's architecture. A *primary content distributor* creates an encrypted version of the library, L_{Enc}, using *per-object keys*, and replicates L_{Enc} to two *secondary content distributors*, each in separate administrative domains. The primary content distributor maintains a *key server*. Each secondary content distributor maintains an *object server* that is distributed over multiple physical machines.

The key server delivers the per-object keys using CPIR; the object servers deliver encrypted objects using ITPIR (§2.3.1). The distinction between key and

[2] The state of the art CPIR implementation is XPIR, which is based on the Ring-LWE cryptosystem. XPIR can process data at 22 Gbps on a machine with 4 physical (and 8 virtual) cores [34], while the CGKS ITPIR implementation in Percy++ [116], based on cheaper XOR operations, can process data at 152 Gbps on comparable hardware.

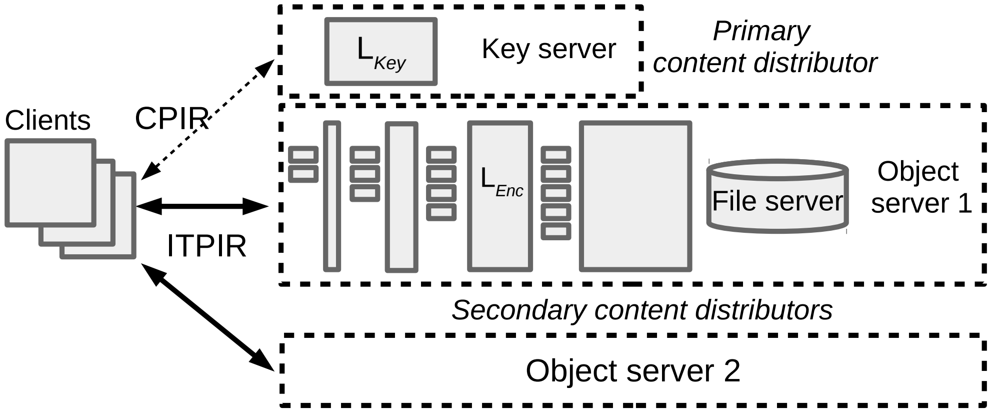

Figure 2.4: Popcorn's architecture. Popcorn uses two servers for ITPIR. Each object server stores all of the columns in the library (Figure 2.5), and is distributed over multiple physical machines.

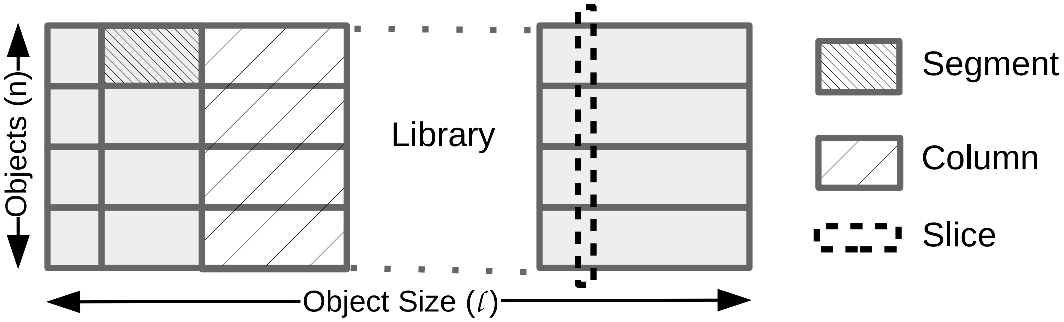

Figure 2.5: Popcorn terminology. Each column is stored by two ITPIR instances (one from each object server). Columns are divided into slices, which are assigned to physical machines.

object servers maps to today's DRM implementations [3, 12, 199], where clients contact two separate servers, one for encrypted video and one for decryption keys.

Media objects are split into *segments*—contiguous pieces of media containing, for example, a few seconds or minutes of a video. Segment sizes vary (§2.3.3). Each object is presumed to have the same decomposition into segments (we revisit this assumption in §2.3.4). The library is partitioned into *columns* (Figure 2.5); a column is a union of corresponding segments, across all objects. Therefore, a column's size is n times that of any segment it contains.

Each column is stored and served by two independent ITPIR *instances* (one for each object server); different instances use separate physical machines. Columns are further sub-divided into *slices*, which are the work units assigned to physical machines. A slice is 1 MB “wide” and n items “high”; we sometimes refer to 1 MB as a *chunk*. Each machine is responsible for one or more slices.

To retrieve an object, the client fetches a decryption key from the key server and the encrypted object from the object servers. The latter step proceeds in two overlapping phases. In the first phase, the client sends, in parallel, a query vector to all machines in both object servers. On receiving a request, a machine adds the query vector to a request queue. Each machine services its queue by: looping over its slices, computing chunk-sized ITPIR replies for every pending request, and pushing the resulting chunks to a file server (one per object server; Figure 2.4) that retains the chunks until they are requested by clients. In the second phase, the client downloads these ITPIR-encoded chunks at the appropriate playback times, and applies Decode (Figure 2.2). This phase overlaps with the server-side generation of replies.

2.3.1 Composing ITPIR and CPIR

As stated earlier, Popcorn combines CPIR and ITPIR: the heavier-weight CPIR, which requires only one server, is used to serve per-object keys, while the lighter-weight ITPIR is used to serve the large encrypted objects. As a result, both keys and objects are served privately (because PIR is applied to them both), CPIR is not a performance bottleneck (because it is used only for small keys), and current controls on content protection are respected (because the plaintext content and keys are stored only at the primary content distributor).

As an alternative to CPIR, the key server could use Symmetric PIR (SPIR) or 1-out-of-n Oblivious Transfer (OT). Section 2.6 discusses these alternatives.

2.3.2 Batching

Popcorn uses the CGKS ITPIR scheme described in §2.1.4, as its inexpensive operations (XORs) keep its computational overhead low (by the standards of PIR). Still, because ITPIR queries are dense—on average, half of the entries are set to 1 (Figure 2.2)—responding to a query requires the machine serving a slice to read from storage and XOR, on average, $n/2$ chunks. This taxes I/O bandwidth, memory bandwidth, and CPU cycles.

To reduce costs, Popcorn's machines, which are oblivious to the content of queries, process queries in *batches*, and perform a single I/O pass over a slice for all of the queries in a batch. Batching thus amortizes I/O overhead and lets Popcorn exploit sequential transfer bandwidth.

Batching also reduces *computational* (not just I/O) overhead by leveraging the observation that the PIR computation required for a batch of requests can be expressed as matrix multiplication ($q \cdot L$ in Figure 2.2 can be replaced by $Q \cdot L$, where Q is a matrix whose rows are query vectors). Previous work [51, 172] (covered by the Percy++ row in Figure 2.3) has used this observation to incorporate sub-cubic algorithms [77, 144] that reduce the total number of operations required by PIR. Popcorn, by contrast, chooses block matrix multiplication [163], which, though it does not affect the total number of operations, leverages cache locality. One can view the resulting access pattern as batching at the CPU-memory interface.

2.3.3 Specializing batching for media delivery

Given the considerations in the previous subsection, Popcorn has an interest in increasing batch sizes (at least up to a point).[3] However, there is a tension between large batch sizes, which seem to require synchronizing clients, and meeting the deadlines of real-time media delivery. Popcorn resolves this tension as follows.

To begin with, each ITPIR instance loops over its assigned column (§2.3) continuously. Since a client can begin playback only after decoding the response for the first column, Popcorn uses a "narrow" first column to keep this initial delay short. Column width, however, increases quickly in Popcorn, making later columns wide. The crucial intuition is that *wide columns imply good batching opportunities*: a batch comprises all requests that reached an ITPIR instance during its previous loop interval, and wider columns imply longer loop intervals.

Figure 2.6 depicts this arrangement. It is inspired by Pyramid Broadcasting (PB) [240] (see also [18, 135]), wherein an object is divided into increasingly-sized pieces, each served on a separate broadcast channel that loops over the piece.

[3] Above a certain batch size, there is no advantage: I/O is no longer a bottleneck, and the CPU benefits of using matrix multiplication stop increasing. However, there is also no disadvantage, so for simplicity, Popcorn does not bound batch sizes.

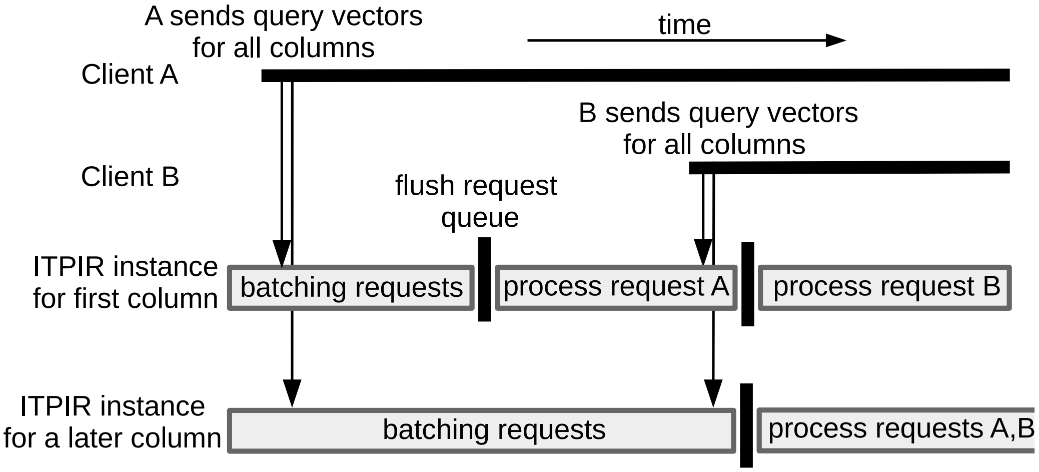
A sends query vectors for all columns
time
Client A
B sends query vectors for all columns
Client B
flush request queue
ITPIR instance for first column
batching requests
process request A
process request B
ITPIR instance for a later column
batching requests
process requests A,B

Figure 2.6: Batching at an object server in Popcorn. Requests to the initial column from two clients A,B are in separate batches as the processing cycle for this column is short. The requests to a later column (sent alongside the requests to the first) can be batched. This arrangement is inspired by Pyramid Broadcasting [240].

Differences are as follows. Whereas PB targets network bandwidth efficiency (and clients must buffer), Popcorn aims to reduce server-side I/O (and the buffer is at the server); one can view Popcorn's arrangement as the I/O subsystem using PB to "broadcast" to the next layer in the pipeline (the XORs). Furthermore, in Popcorn, a server's work (the XORs) depends on the number of clients (unlike in a broadcast setting). Finally, in Popcorn, each instance is distributed over multiple physical machines. These differences lead to a design and analysis that owe a debt to PB but are specific to our context.

Details. We start with two simplifying assumptions, which we revisit later: that a single ITPIR instance is handled by a single machine, and that there is no network delay or loss. Define an *instance processing cycle* as the duration of one iteration of an instance's loop. Within this cycle, an instance traverses each slice in turn, performing Answer for all queries that arrived during the prior cycle.

We want all clients to experience smooth playback. To this end, suppose that we are willing to impose startup delay d. Suppose further that $T_1 \leq d - \epsilon$, where T_1 denotes the processing cycle for the first instance, and ϵ is the time for the instance to handle a single slice. Likewise, define T_i as the processing cycle for the

ith instance ($i > 1$), and suppose that for all such instances, $T_i \leq d - \epsilon + \sum_{j=1}^{i-1} t_j$, where t_j is the playback time of segment j.

Under these conditions, we claim that any client, *regardless of when it joins*, experiences smooth playback. Why? Consider only instance 1: in the worst case, a client initiates consumption just after instance 1 begins its processing cycle. The client cannot download until the current processing cycle has terminated (which takes time T_1) and the first slice of the next cycle is processed (for an additional ϵ). Smooth playback simply requires the overall delay ($T_1 + \epsilon$) to be less than d, matching our conditions. Once playback begins, the client has t_1 additional time before it needs the second segment. Generalizing, in the worst case for instance i (i.e., the client's initial request arrives just as a processing cycle begins), as long as T_i is no larger than $d - \epsilon + \sum_{j=1}^{i-1} t_j$ (which is exactly what our conditions guarantee), then the first slice of the ith instance will be ready, and playback will be smooth.

But how should the $\{t_i\}$ be set? Recall that, for more effective batching, Popcorn needs segment widths to increase: we are then seeking the maximum t_i for each instance i .

Let μ be the playback rate, P_i the rate at which XOR operations are processed by the ith instance, R_i the I/O bandwidth available to the instance, and b_i the batch size (the number of requests accumulated in a cycle of time T_i). To upper-bound t_i, we match load to capacity, for both I/O and CPU. For I/O, the column's data (n segments, each of size $t_i \cdot \mu$) is upper-bounded by the amount of data that the instance can read in one cycle: $t_i \cdot \mu \cdot n \leq T_i \cdot R_i$. For CPU, the picture is similar, except that the total work scales with b_i, the number of clients being served: $t_i \cdot \mu \cdot n \cdot b_i \leq T_i \cdot P_i$. These inequalities lead to:

$$t_i \leq T_i \cdot \left(\frac{\min\{R_i,\ P_i/b_i\}}{\mu \cdot n} \right).$$

Assume that for all i, $\min\{R_i,\ P_i/b_i\} \geq \mu \cdot n$ (we will arrange for this in "Provisioning," below). Then, the foregoing bounds (on the $\{T_i\}$ and on load) imply that for all i, we can set:

$$t_i = T_i = 2^{i-1} \cdot (d - \epsilon)$$

(see Appendix A). Note that the $\{t_i\}$ increase exponentially in size, as desired. In particular, approximately half of the file is covered by the final segment.

Provisioning is driven by the earlier assumption that $\min\{R_i, P_i/b_i\} \geq \mu \cdot n$ for all i. To meet the requirements on R_i and P_i, Popcorn uses multiple machines per instance and aggregates their resources, by striping slices across them. If r_i is the per-machine I/O bandwidth for the machines used for the ith instance, then the I/O for instance i can be handled with $R_i/r_i = \mu \cdot n/r_i$ machines. P_i, the XOR processing throughput for instance i, increases with i because so does the batch size b_i; specifically, if λ is the overall rate at which clients initiate requests for objects, then $b_i = \lambda T_i$. Moreover, the per-machine XOR processing throughput for the ith instance, $p_i(\cdot)$, is a function of the batch size because cache locality in block matrix multiplication (§2.3.2) (and hence throughput) improves with a a bigger batch size. Thus, the task of processing the XOR operations for instance i can be handled by $P_i/p_i(b_i) = \mu \cdot n \cdot b_i/p_i(b_i)$ machines.

To account for the striping, we need to modify the earlier analysis of startup delay, smooth playback, etc.: if resources from k_i machines are aggregated for the ith instance, then each machine takes $\epsilon \cdot k_i$ time instead of ϵ to handle a slice. As a result, the inequality $T_i \leq d - \epsilon + \sum_{j=1}^{i-1} t_j$ becomes $T_i \leq d - \epsilon \cdot k_i + \sum_{j=1}^{i-1} t_j$, and both the $\{T_i\}$ and $\{t_i\}$ are computed accordingly.[4]

The total number of machines required, across all I instances, equals: $\mu \cdot n \cdot \sum_{i=1}^{I} \max\{1/r_i, \lambda T_i/p_i(\lambda T_i)\}$. Notice that if the max is controlled by the first term, then the given instance is bottlenecked by I/O (and the CPU resource is sometimes idle); if by the second, then the instance is bottlenecked by CPU work (and the I/O resource is sometimes idle). Later (§2.5.1) we will obtain estimates empirically for r_i and $p_i(\cdot)$.

Popcorn must also provision for the file server machines (§2.3). The file server requires the buffer space for each instance to equal the number of requests in service times the size of a segment, i.e., $\sum_{i=1}^{I} b_i \cdot (t_i \cdot \mu)$. The file server also requires I/O bandwidth equal to the rate at which reply data is produced and consumed: $2 \cdot \sum_{i=1}^{I} b_i \cdot \mu$ (assuming $t_i = T_i$).

Finally, we have been assuming no burstiness or delay in the network. To account for network fluctuation, we must allow for clients to build up a playback buffer, of some time length β. To this end, T_i should be upper-bounded by $d - \epsilon \cdot k_i - \beta + \sum_{j=1}^{i-1} t_j$, and the $\{t_i\}$ computed to be consistent with T_i.

[4] The computation must resolve a circular dependency as T_i is expressed in terms of k_i, which itself depends on the segment size, with a bigger segment requiring more machines. We resolve this circularity by repeating the process of speculatively setting a k_i, calculating T_i, and then refining the speculated value of k_i using the obtained value of T_i.

Discussion. To understand the savings and amortization from Popcorn's batching, consider a naive batching scheme, in which time is divided into *epochs* of length T_{epoch}. Let a *cohort* denote the set of clients who initiate a request (for the first chunk of a media file) in an epoch. Then, the entire cohort moves through the slices, as it were, together. Each cohort needs enough machines to meet two requirements: (a) $\mu \cdot n$ I/O bandwidth, and (b) $\mu \cdot n \cdot \lambda \cdot T_{\text{epoch}}$ XOR processing throughput (here $\lambda \cdot T_{\text{epoch}}$ is the cohort's batch size). If $H = T / T_{\text{epoch}}$ is the total number of cohorts (where T is the total playback time), then the total number of machines is $\mu \cdot n \cdot \sum_{i=1}^{H} \max\{1/r, \lambda \cdot T_{\text{epoch}} / p(\lambda \cdot T_{\text{epoch}})\}$, where r is the per-machine I/O bandwidth, and $p(\lambda \cdot T_{\text{epoch}})$ is per-machine XOR processing throughput for a batch size of $\lambda \cdot T_{\text{epoch}}$. Here, T_{epoch} must be upper-bounded by $d - \epsilon \cdot k - \beta$ to meet the startup delay requirements, where k is the number of machines for a cohort.

To compare the cohort batching scheme to Popcorn, we make the simplifying and optimistic assumption that both schemes use machines that make the two terms of the respective maxes equal, so that no resources are idle (we will revisit this assumption in §2.5.2 and §2.5.4). Then, the total I/O bandwidth required by the cohort scheme is $H \cdot \mu \cdot n$, which is considerably larger than what Popcorn needs ($I \cdot \mu \cdot n$, where $I \ll H$).

In terms of computational resources, the cohort scheme needs $\mu \cdot n \cdot \lambda \cdot T / p(\lambda \cdot T_{\text{epoch}}) = \mu \cdot n \cdot \lambda \cdot \sum_{i=1}^{I} T_i / p(\lambda \cdot T_{\text{epoch}})$ machines; Popcorn requires instead $\mu \cdot n \cdot \lambda \cdot \sum_{i=1}^{I} T_i / p_i(\lambda \cdot T_i)$ machines. Neither scheme is the clear-cut winner; however, if we assume that $p(\cdot) = p_i(\cdot)$ for all i, then Popcorn has lower computational demands, because (a) $T_i \approx 2^{i-1} \cdot T_{\text{epoch}}$ (by our earlier analysis) and (b) $p(\cdot)$ is monotonically increasing. In essence, Popcorn has larger batches, so (holding machine type configuration constant) the benefit of locality is more pronounced (§2.3.2), lowering Popcorn's computational requirements relative to the naive batching scheme.

2.3.4 Handling variable-sized objects

The design has so far assumed equally sized objects. A naive solution would be to pad all objects to the size of the longest one. However, this would, for Netflix, cause a 4× increase in network transfers: the average movie is approximately 1.5 hours while the longest is 6, and clients would have to download the padding in full (doing otherwise would reveal the true object size).

Popcorn's solution instead chooses a representative object O_{avg} from the library (for example, the object closest to the average media length) and pads smaller objects to that size. Longer objects, up to a cutoff, are compressed down to O_{avg}'s size, by reducing the bitrate;[5] the longest objects are split into several files.

We note that reducing the bitrate is likely to be tolerable, as variations of up to 30% (roughly) in video bitrate have a limited impact on user satisfaction [97, 154, 220]. (Other factors, such as playback interruptions and startup times, instead have substantial impact.) Objects that cannot be tolerably compressed must be divided up (as in other systems [86, 133]). However, the client would then have to download each division as if it were a separate movie, which means delaying consumption or downloading far ahead of time (if the separate divisions were downloaded all at once, then an attacker could guess that a longer object is being consumed).

The Netflix catalog [1] indicates that the majority of movies have a similar size: 85% of the objects are between 60 and 120 minutes, with the majority clustered around the average movie length of 92 minutes. Movies between 92 and 120 minutes will require 23% compression in the worst case and 10% on average; similarly, the padding for objects between 60 and 92 minutes will be small to moderate. The impact of objects at either extreme will be limited: 8% of the movies are shorter than 60 minutes, and will require significant padding; 5% are between 120 and 135 minutes, making them candidates for aggressive compression (32% in the worst case and 27% on average) though potentially at the cost of lowering user satisfaction; and 2% are over 135 minutes, making them candidates for splitting. We think that splitting is not a huge limitation, because we hypothesize that people usually plan ahead to watch long movies.

2.4 Implementation

Our prototype implements the design in Section 2.3, except for large file splitting (§2.3.4). It leverages existing PIR implementations: the key server uses the XPIR [34] implementation of the CPIR protocol in Figure 2.1. For the object servers,

[5] Regardless of an object's bitrate, a client must issue chunk download requests at a constant rate (e.g., one request every 1 MB/μ seconds, where 1 MB is a chunk's size and μ is O_{avg}'s bitrate); otherwise, chunk download patterns would leak information.

Popcorn is affordable when it serves large media files to many concurrent clients.	§2.5.2
Popcorn's per-request dollar cost is $3.87\times$ of a system without privacy for workloads with $\geq$10K concurrent clients.	§2.5.3
Popcorn integrates well with existing web technology. It can play DRM-encoded media within modern web browsers.	§2.5.6

Figure 2.7: Summary of Popcorn's evaluation results.

we borrow the CGKS ITPIR implementation of Percy++ [116][6] and modify it to support the techniques in Section 2.3. The total server-side code is 11K lines of C++. We implement two versions of the client-side library: one in C++ (2500 lines), which we use for experiments (§2.5.2), and one in JavaScript (500 lines), which we use to show compatibility with modern web browsers (§2.5.6).

2.5 Evaluation

Our evaluation answers the following questions:

1. When is Popcorn affordable?
2. What is the price of Popcorn's privacy guarantees?
3. Can we use Popcorn to watch a movie encoded using an existing DRM scheme on a modern web browser?

Figure 2.7 summarizes our evaluation results.

Method and setup. We compare Popcorn to three baselines: *NoPriv*, *BaselinePIR*, and *BaselinePIR*++. NoPriv serves object chunks from an Apache web server, modeling media delivery systems that use HTTP caching at CDN edge servers [32]. BaselinePIR is a modified version of Percy++ [116] CGKS: the servers store the library L as slices and process ITPIR queries directed at them. This is essentially Popcorn without the techniques of §2.3. BaselinePIR++ additionally batches requests using cohort batching (§2.3.3) to reduce both I/O and CPU costs. For all PIR sys-

[6] Percy++'s CGKS ITPIR implementation is one of the fastest implementations for two-server ITPIR. An alternative is the CGKS implementation from RAID-PIR [86] (§2.6).

	type	vCPUs	RAM (GB)	SSDs (# × GB)	cost/hr
c3.8xl	1	32	60	2 × 320	\$0.6281
i2.4xl	2	16	122	4 × 800	\$0.8451
i2.8xl	3	32	244	8 × 800	\$1.6902

Figure 2.8: Hourly cost of reserved Amazon EC2 machines used in our experiments. Machines starting with "c" are compute-optimized; those starting with "i" are I/O-optimized.

tems, we experiment with one object server and multiply the measurements by two (to reduce the financial cost of our experimental evaluation).

Our workload is modeled on existing media delivery services [227]: clients arrive according to a Poisson process (e.g., $C = 10K$ clients arrive in $T = 90$ minutes). All clients in NoPriv request the same (average-size) object, giving this baseline the maximum benefit of server-side caching. The server's work in Popcorn, BaselinePIR, and BaselinePIR++ is oblivious to the request distribution (we select a Zipfian distribution with $\theta = 0.8$).

For the four systems, we measure server- and client-side resource usage in terms of CPU time (by instrumenting code with `clock()`), I/O transfers and storage (using `iostat`), and network transfers (via `/proc/net/dev`).

Our experimental testbed is a single availability zone within Amazon's EC2, and is described in Figure 2.8.

2.5.1 Provisioning resources using microbenchmarks

Popcorn. Machine provisioning for Popcorn involves two steps: (1) benchmarking the basic operations (details in Figure 2.9), and (2) combining the results with the provisioning analysis in §2.3.3.

Consider, for example, provisioning the first ITPIR instance of a Popcorn object server for a Netflix-like workload: $C = 10{,}000$ clients streaming from a library of $n = 8192$ media files with average playing time of $T = 90$ minutes, playback rate of $\mu = 4$ Mbps, and startup delay of $d = 15$ seconds.[7] The processing cycle of this

[7] We think that 15 seconds of delay before playing a long video is tolerable. During this time the server could display a generic advertisement or public service announcement (existing services commonly display 15 or 30 second advertisements [30]).

	Throughput (Gbps)		
	c3.8xl	i2.4xl	i2.8xl
Sequential read	6.4	12.6	23.3
Random mixed rw	2.1	8.0	16.0
block matrix multiplication	488–4968	488–2512	432–4608

Figure 2.9: Throughput of basic operations in Popcorn—reading a column slice (§2.3.3), reading and writing 1 MB sized chunks, and computing block matrix multiplication on a slice (§2.3.2)—on machines listed in Figure 2.8. The latter value depends on the size of the query matrix (§2.3.2, §2.3.3), so we report a range: from a query matrix consisting of a single query vector to one that contains 4096 query vectors.

instance must be $T_1 \leq d - \epsilon \cdot k_1$ (§2.3.3). For our example, $\epsilon = 2$ (the time to process or consume a 1 MB chunk at $\mu = 4\,\text{Mbps}$), and we speculatively set $k_1 = 3$, which gives $T_1 \leq 15 - 2 \cdot 3 = 9$ seconds. Thus, the instance is given a segment of $t_1 = T_1 = 9$ seconds and has a batch size of $b_1 = (C/T) \cdot T_1 = 17$. Furthermore, it requires storage capacity of $n \cdot t_1 \cdot \mu = 36\,\text{GB}$, read bandwidth $R_1 = n \cdot \mu = 32\,\text{Gbps}$, and XOR processing throughput $P_1 = b_1 \cdot n \cdot \mu = 544\,\text{Gbps}$.

Our microbenchmarks (Figure 2.9) indicate that these requirements can be met by three i2.4xl machines. If the microbenchmarks had indicated a different number, then, as described in §2.3.3, we would have had to adjust k_1 (which was speculatively set) and repeat the provisioning process described above.

BaselinePIR. To use the fewest possible machines, we stripe the approximately 21 TB library of our Netflix-like workload across machines with highest storage capacity (that is, i2.8xl in our testbed).

To reduce the financial cost of our experimental evaluation, we measure the number of requests that can be serviced by this setup, along with each request's resource consumption, and extrapolate the results to workloads with a larger number of requests (e.g., to support 2× concurrent clients, we double resource costs).

BaselinePIR++. As in Popcorn, we use the microbenchmarks in Figure 2.9 and the provisioning analysis for the cohort batching scheme (§2.3.3).

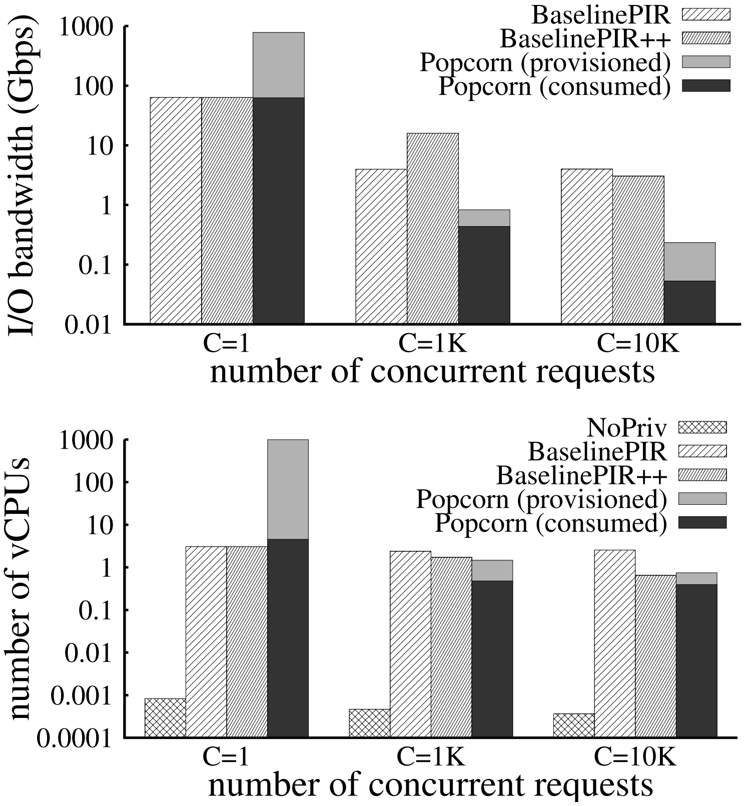

Figure 2.10: Per-request server-side resource use (log-scaled) of Popcorn and the baselines with varying concurrent requests C. If I/O is the bottleneck, there are idle CPU cycles and vice versa (§2.3.3). For Popcorn, we depict both the provisioned and consumed resources; for the baselines, we depict only the latter. We do not depict I/O usage for NoPriv as it is always zero (see text).

2.5.2 Per-request overheads of Popcorn

To understand when Popcorn is affordable, we run experiments varying the number of concurrent requests (C); the number of objects (n); and the playing time of objects (T). We find that Popcorn incurs modest costs when the library size is moderate (≈8K media files), object sizes are large (≈90 minutes), and there are many concurrent clients (≥10,000). Fortunately, these settings are consistent with the workloads of Netflix-like systems (§2.7).

Before proceeding, we note that Popcorn's provisioning method can leave resources idle (§2.3.3), so we report both the consumed and provisioned resources.

We focus on the consumed resources in this subsection and account for the idle resources in the next subsection.

Overhead versus number of concurrent requests. We run Popcorn and its baselines with $C = \{1, 1\text{K}, 10\text{K}\}$ while keeping $n = 8192$, $T = 90$ minutes, $\mu = 4$ Mbps, and $d = 15$ seconds. Figure 2.10 summarizes the per-request server-side resource costs.

I/O overheads. When $C = 1$, the I/O bandwidth Popcorn consumes matches that of BaselinePIR and BaselinePIR++, as there is no opportunity to batch requests. However, as the request rate increases, batching lets Popcorn amortize its I/O transfers (§2.3.3): the per-request amortized I/O bandwidth decreases from ≈ 63 Gbps (for $C = 1$) to 53 Mbps (for $C = 10K$), a reduction of $1190\times$. Surprisingly, concurrent requests, by hitting the file system cache, also reduce BaselinePIR's per-request I/O bandwidth (by $16\times$). As expected, BaselinePIR++'s per-request I/O bandwidth reduces by the cohort batch size. Finally, there are no I/O transfers in NoPriv as all requests hit the same (cached) object.

CPU overheads. For a single request, Popcorn consumes 50% more CPU than BaselinePIR, as the overhead of parallelizing block matrix multiplication (over multiple cores) in Popcorn (§2.3.2) is charged to a single request. As the number of concurrent requests increases, Popcorn's CPU overheads decrease; the per-request CPU consumption decreases by $\approx 11\times$ when the number of concurrent requests increases from 1 to 10,000. We hypothesize that this stems from the increase in cache locality from block matrix multiplication over bigger batch sizes.[8] Furthermore, the 36 minutes of per-request CPU time for $C = 10\text{K}$ matches the performance of the matrix multiplication microbenchmark (42 TB of data processed in 36 minutes gives a throughput of 159 Gbps for a single CPU, consistent with the throughputs reported in Figure 2.9).

Popcorn's per-request CPU consumption is much higher than NoPriv ($1080\times$ for $C = 10\text{K}$): for a single object, the Apache web server in NoPriv serves 1 MB chunks and requires almost no server-side processing, whereas Popcorn XORs n objects on average.

[8] In a separate experiment, we measured the percentage of cache misses for block matrix multiplication (§2.3.2) using CPU performance counters, and found that it reduces from 48% for a query matrix with a single request to less than 2% for a query matrix with 2^{10} requests.

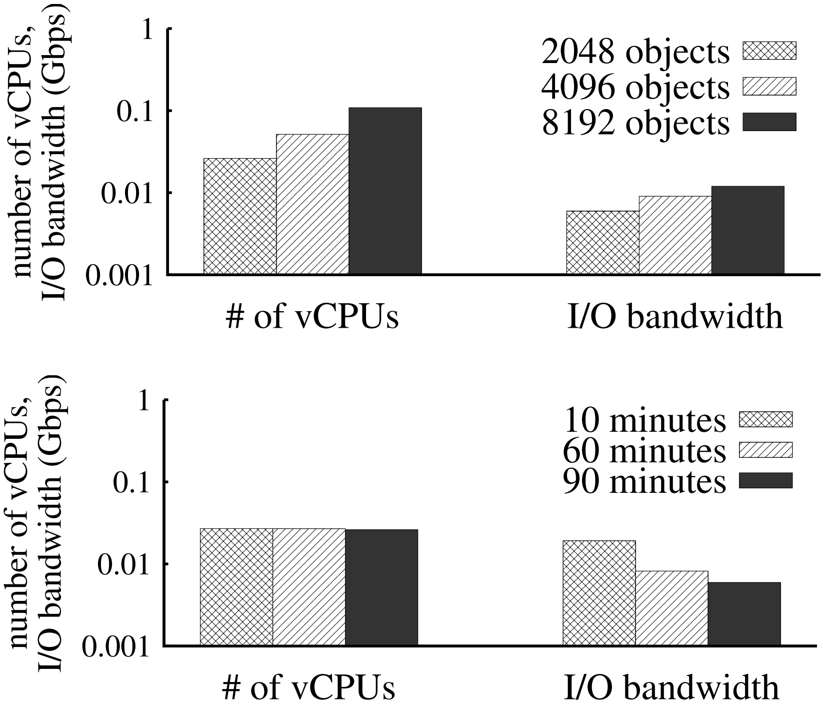

Figure 2.11: Popcorn per-request resource use (log-scaled) as a function of the number (top) and length (bottom) of objects.

Network and storage overheads (not depicted in the figures). Popcorn, BaselinePIR, and BaselinePIR++, incur a two-fold network overhead over NoPriv because clients download from two servers. With respect to storage, each instance of an object server in Popcorn needs buffer space equal to its segment size times its batch size (§2.3.3). Across all instances, this equals ≈15.4 TB, or ≈1.6 GB per concurrent request, which is 0.6× the size of an object.

Overhead versus number of objects. In Figure 2.11(a), we change the size of the library ($n = \{2048, 4096, 8192\}$) while keeping the other parameters fixed ($C = 10$K, $T = 90$ min, $\mu = 1$ Mbps,[9] and $d = 15$ seconds). As expected, Popcorn's per-request CPU and I/O bandwidth consumption, even though amortized, is proportional to n. Network downloads and server-side storage overheads (not shown) do not change with n.

[9] To reduce the financial cost of EC2 experiments, this and subsequent experiments set $\mu = 1$ Mbps instead of 4 Mbps. The change scales down the experiments; the qualitative results are unaffected.

Overhead versus playing time of objects. In Figure 2.11(b), we change the playing time of objects ($T = \{10, 60, 90\}$ minutes) while keeping the other parameters fixed ($n = 2048$, $\mu = 1$ Mbps, $d = 15$ seconds, and $C = 10$K). As T increases, the per-request CPU consumption is unaffected. Also, with increasing T, the per-request I/O consumption decreases; on the other hand, idle I/O bandwidth (not depicted in the figure) increases (§2.3.3).

Overheads of the key server. Recall that Popcorn uses XPIR [34] as its CPIR implementation (§2.4). Since XPIR does not batch requests, the per-request overheads of the key server depend only on the number of keys (and not on the number of concurrent requests C). We use a single machine of type c3.8xl for the key server. For a library with 8192 keys, it takes three seconds of server-side CPU time to privately retrieve a key; there are no I/O transfers because the 128 KB library fits in memory. Thus, as expected, the key server is not a performance bottleneck for Popcorn. Moreover, the end-to-end time to retrieve a key is much less than the startup delay of d =15 seconds.

Client-side overheads. Compared to NoPriv, Popcorn's client consumes additional CPU and network bandwidth (because it has to generate and decode PIR queries, and download content from two object servers). For example, for $n = 8192$ objects, $T = 90$ minutes, and $\mu = 4$ Mbps, we find that Popcorn's client (run on a single vCPU of c3.8xl type machine) consumes 10 CPU seconds (compared to NoPriv's 1.7 CPU seconds), and 25 MB of network upload bandwidth (compared to NoPriv's 11 MB).

2.5.3 Dollar-cost analysis

The previous subsection showed that Popcorn significantly reduces CPU and I/O consumption over the baseline PIR systems, at least for large objects and high load. These improvements provide the foundation for achieving privacy at low cost, a cost that we now quantify.

Method. We use the pricing model of Amazon EC2 (Figure 2.8) to estimate the per-request machine cost, and the pricing model of CDNs ($0.006 per GB) [202] to compute per-request network cost. We choose these pricing models because

	experimental configuration				per-request costs ($)		
	#reqs	#1	#2	#3	machine	network	total
NoPriv	10K	–	–	–	–	0.016	0.016
Popcorn	1	2	60	0	77.943	0.032	77.975
Popcorn	1K	17	50	4	0.09	0.032	0.122
Popcorn	10K	185	32	32	0.03	0.032	0.062

Figure 2.12: Estimated per-request dollar cost for NoPriv and Popcorn. #1, #2, and #3 refer to the type of AWS EC2 machines from Figure 2.8.

they are public—though, in an actual deployment, a service could receive wholesale, bulk, or negotiated prices. We use a Netflix-like workload in our calculations: $n = 8192$ media files, $T = 90$ minutes, $\mu = 4$ Mbps with varying number of concurrent clients. Figure 2.12 summarizes our results. We find that Popcorn's per-request cost is within a small multiple of NoPriv for a workload with $C = 10$K concurrent clients.

NoPriv. To give NoPriv the maximum benefit, we disregard its machine cost. The per-request cost is then determined solely by the network transfer cost, and is $\approx$ \$0.016 (i.e., 90 minutes $\times$ 4 Mbps $\times$ \$0.006/GB).

Popcorn. We provision EC2 machines as described in §2.5.1 and §2.3.3. The total per-request cost is derived by combining (1) the per-request machine cost, computed by dividing total machine cost by the total number of requests, and (2) the per-request network cost. This method charges Popcorn for both consumed and idle resources (Figure 2.10). For the Netflix-like library and $C = 10$K, the per-request cost is \$0.062 (the per-request machine cost is \$0.03; the per-request network cost is \$0.032).[10] Popcorn thus increases dollar cost 3.87× over NoPriv, in line with our initial affordability requirement (§1). Importantly, Popcorn's low cost is premised on many clients accessing the system concurrently: the per-request machine cost decreases with the number of concurrent clients. It is \$78 for $C = 1$ and \$0.09 for $C = 1$K.

[10] The network cost can be reduced for a pricing model in which network transfers between (ITPIR) servers is cheaper than server to client transfers, by using the techniques of Riffle [162, Section 4.4] (§2.6).

system	description
XPIR [34]	fastest CPIR implementation
XPIR++	XPIR with naive batching (§2.3.2)
BaselinePIR	XPIR composed with CGKS ITPIR (§2.3.1)
BaselinePIR++	BaselinePIR with naive batching (§2.3.2)
Popcorn	§2.3.1 + §2.3.2 + §2.3.3

Figure 2.13: Comparison points. "Naive batching" refers to an instantiation of batching, as described in Section 2.3.2, with the cohort batching scheme described in Section 2.3.3.

BaselinePIR and BaselinePIR++. Since we might have provisioned these systems wastefully, we do not estimate their dollar cost using the machine-based pricing model, which charges for both the consumed and idle resources. Instead, we use a per-resource pricing model to estimate the dollar cost of these systems, as described next.

2.5.4 Further comparisons

In this subsection, we estimate the dollar cost of BaselinePIR, BaselinePIR++, XPIR, and XPIR++, a hypothetical extension to XPIR that uses cohort batching (§2.3.2) to reduce I/O costs (but does not use matrix multiplication). Figure 2.13 summarizes these alternatives.

The estimates for BaselinePIR, BaselinePIR++, and Popcorn are based on experimental data from §2.5.2; for XPIR and XPIR++, we calculate CPU resource consumption using XPIR's reported performance and I/O bandwidth consumption from the expression $2 \cdot (n \cdot \mu) / b_{\text{cohort}}$ (the factor of two is due to XPIR's preprocessed library being twice the size of the original [34]).

We compare these systems for the Netflix-like workload of §2.5.3. We set the startup delay d to 15 seconds, except for the systems using cohort batching scheme, for which we vary d.

We use a per-resource pricing model (derived in Appendix B) based on Amazon EC2's machine cost (Figure 2.8) and on the network cost of CDNs [202]. Our model charges CPU at \$0.0076/hour, I/O bandwidth at \$0.042/Gbps-hour, and network transfers at \$0.006 per GB. Multiplied by each system's consumption of the corresponding resources, these values determine the per-request dollar cost (Figure 2.14).

	# vCPUs	I/O bandwidth (Gbps)	Dollar cost relative to NoPriv
X [34]/X++ (C=1)	11.6	64	265×
X++ (C=1K)	11.6	26.6	118×
X++ (C=1K, d=60)	11.6	5.96	37×
X++ (C=1K, d=600)	11.6	0.58	16×
X++ (C=10K)	11.6	2.66	24×
X++ (C=10K, d=60)	11.6	0.59	16×
X++ (C=10K, d=600)	11.6	0.058	13.5×
B/B++ (C=1)	3.1	64	256×
B (C=1K)	2.4	4	19×
B (C=10K)	2.5	4	19×
B++ (C=1K)	1.7	16	66×
B++ (C=1K, d=60)	1.26	9.15	39×
B++ (C=1K, d=600)	0.49	0.54	4.5×
B++ (C=10K)	0.65	3	14×
B++ (C=10K, d=60)	0.49	0.59	4.7×
B++ (C=10K, d=600)	0.41	0.058	2.5×
P (C=1)	4.6–992	63–781	253×–4873×
P (C=1K)	0.5–1.47	0.43–0.83	4×–7.6×
P (C=10K)	0.4–0.74	0.053–0.23	2.5×–3.87×

Figure 2.14: Per-request resource consumption and estimated dollar-cost of XPIR (X), XPIR++ (X++), BaselinePIR (B), BaselinePIR++ (B++), and Popcorn (P). Network transfers are not shown; they are 5× NoPriv for X and X++, and 2× NoPriv for the other systems. For Popcorn, we present a range: the smallest value considers only the consumed resources, while largest value includes both consumed and idle resources. Startup delay d is 15 seconds unless specified otherwise.

- The costs of XPIR are high (265× NoPriv), though adding a naive batching scheme (XPIR++) significantly reduces them (by ≈11× for $C = 10$K, $d = 15$).
- Using ITPIR for object delivery (in conjunction with CPIR (§2.3.1)) reduces the costs further (by $\approx 2\times$ for $C = 10$K, $d = 15$). The disadvantage is that ITPIR requires non-colluding servers.
- Increasing the startup delay (and thus the batch size of the cohort) can further reduce costs. For example, increasing d from 15 to 60 seconds reduces

costs by $3\times$ (a reduction from $14\times$ NoPriv to $4.7\times$ NoPriv).

- BaselinePIR++ matches the cost of Popcorn (when $C = 10$K) but requires a $40\times$ higher startup delay ($d = 10$ minutes in BaselinePIR++ vs. 15 seconds for Popcorn).

2.5.5 Discussion

The analysis presented in the previous two subsections estimates that adding privacy to a Netflix-like service would increase resource costs by 3.87 times. However, it does not indicate the broader economic impact on the service, which depends on many factors including resource costs. On the one hand, the increased resource costs may drive up the subscription fee and reduce the number of subscribers. On the other hand, the added privacy could encourage the addition of sensitive content (adult movies, etc.) and drive up the number of subscribers. Therefore, it may be difficult to predict, in general, revenue with an increase in resource costs, and in particular, effect on subscription fee seen by customers. To understand the broad economic impact a detailed analysis is required, which is beyond the scope of this dissertation.

2.5.6 Compatibility study of Popcorn

To verify Popcorn's compatibility with modern Web browsers and DRM technology, we implemented a Popcorn client in JavaScript and used it to watch short videos in the WebM format [25] (protected using WebM Encryption [26]). Our prototype works on Chrome (version 45.0.2454), and makes use of the HTML5 video tag and extensions: the decoded ITPIR content is passed into the Media Source Extension interface, which forwards media chunks to the video player; the decoded CPIR response is also passed into the Encrypted Media Extension interface, which decrypts the protected content.

2.6 Related Work

Alternatives to PIR for privacy. *Obfuscation* [49, 99, 203] protects clients' privacy by cloaking traffic with dummy requests. This approach requires less processing than PIR at clients and servers, but significantly higher network cost: matching

PIR's degree of privacy (the number of objects among which a request is hidden) would require downloading the entire library.

Rather than the content being consumed, *anonymity* hides the identity of the consumer [94, 165]. This could be used to hide metadata (login times, download frequency, etc.), which is complementary to PIR. However, anonymity-based solutions can reveal access patterns that, combined with other background information, may disclose a user's media consumption [187].

Oblivious RAM (ORAM) [119, 175, 178, 225] algorithms conceal a client's access patterns from a storage server. Similarly, *searchable symmetric encryption* (SSE) (surveyed in [68, 69]) offers yet another solution for private data retrieval from a remote database. However, these solutions target a setup where the client outsources its encrypted data to a server.

Recent results [149, 195] enhance the above setup: they let clients privately retrieve data from a remote database owned by a *different* entity. Unlike PIR, these protocols allow for a controlled amount of leakage in the form of data-access and query patterns. Unlike us, they assume that the server does not collude with clients (e.g., in Popcorn the server can pretend to be a new customer of the streaming service). If the server can collude with a client, it can issue queries for each media file in the system, monitor access patterns, and decode all other clients' queries.

Improving the performance of PIR. The computational challenges of PIR have been obvious since its introduction, and have since been mitigated in several ways.

Distributing the work, either by moving it to the cloud or by dividing it among clients [88, 177, 194], reduces latency but not the total computational burden.

GPUs [83, 181] and cheaper cryptographic operations [34, 36, 101, 232, 254] have reduced the computational load of CPIR, refuting the notion [221] that CPIR is likely to be more expensive than the naive solution of transferring the entire library. However, the single request cost for media delivery in XPIR [34], the fastest system employing these techniques, is still higher than desirable (see §2.5.4 and Figure 2.14 for a comparison with Popcorn).

Another path to better performance is to limit the privacy guarantees to only a portion of the library [190, 191, 242]. For example, bbPIR [242] allows users of libraries that can be thought of as a matrix to specify a submatrix (called a bounding box) from which bits can be privately retrieved using CPIR. This approach can be useful for efficiently implementing privacy-preserving location-based services: the larger the bounding box, the higher the privacy, but also the higher the processing and network costs.

Perhaps the most direct way to reduce the overhead of PIR is to genuinely reduce the work that servers need to perform. Lueks and Goldberg [172], building on earlier theoretical work by Beimel et al. [51] and Ishai et al. [148], show that one can achieve sub-linear server-side computation by efficiently processing batches of requests from multiple clients. Popcorn is inspired by this work: it uses batching at multiple stages of its protocol, but tailored for media delivery. Another recent system, RAID-PIR [86], based on the implementation of upPIR [67], reduces server-side work, first, by storing and processing only a *fraction* of the library at each ITPIR server and, second, by encoding multiple requests from the same client in a single query. Popcorn's performance could potentially benefit from these techniques, but only when using more than two servers, or when clients issue multiple simultaneous requests. Currently, Popcorn assumes exactly two servers and that clients request objects sequentially.

Finally, performance can be improved with dedicated hardware [44, 145, 171, 222, 247], at the price of having to trust its manufacturer: a client can connect to a secure coprocessor that (obliviously to the server hosting the library) retrieves and delivers the requested object.

A large body of literature focuses on instead reducing the communication overhead of PIR [110, 192]. Unlike Popcorn, these protocols target an environment in which $n \gg \ell$. In that context, Devet et al. [89] propose a technique that, like Popcorn, composes CPIR and ITPIR. Unlike Popcorn, the composition is hierarchical (ITPIR selects a sub-library, and iterations of CPIR select an object) and minimizes communication costs.

In very recent work, Riffle [162], like Popcorn, targets the case $\ell \gg n$. Unlike Popcorn, Riffle focuses on peer-to-peer file transfers (as opposed to centralized streaming media). Riffle uses ITPIR, with $k > 2$, and focuses on reducing server-to-client network transfers (§2.1.4), by adding server-to-server transfers; this could potentially be composed with Popcorn.

Protecting library content in PIR. The tension between ITPIR and content protection has been noted before. Gertner et al. [113] introduce the problem and propose two solutions, both of which, at a high level, protect the content by storing at untrusted servers independent random data (e.g., two servers store random data that XORs to the library content). Goldberg's ITPIR protocol [117] has a similar protection property as [113], but it uses fewer servers. Huang et al. [139] protect library content kept at untrusted servers by first encrypting it, and then using a threshold signature scheme [87] to serve keys for the encrypted object. In all the above schemes, the library content can be disclosed if more than a threshold of untrusted servers collude. By composing CPIR and ITPIR (§2.3.1), Popcorn instead keeps content protection collusion-proof.

Symmetric PIR (SPIR) schemes add an additional facet to content protection by preventing dishonest clients from learning information about the content of a database beyond what is contained in the records they retrieved [114]. Popcorn currently assumes an honest client (§2.1.1) and thus does not use SPIR to privately download keys from the key server; however, it can reduce that trust by transforming its CPIR protocol into an SPIR protocol [91, 186].

1-out-of-N oblivious transfer (OT) [60, 186] provides the same content protection property as SPIR but, unlike SPIR, can have network overhead linear in the size of the library. In our experiments, this overhead would not be costly: WebM Encryption (§2.5.6) sets our keys to 128 bits, which, for $n{=}8192$ objects, yields a library of only 128 KB. However, the linear overhead can in general be large (e.g., if the key server embeds keys within DRM licenses); for this reason, Popcorn's key server does not use OT.

Handling variable-sized objects in PIR. A naive solution is to pad every object to the size of the longest, and download (the equivalent of) the longest object from each server. Prior work [86, 133] avoids this solution by (a) concatenating small objects (e.g., a few objects form one row of the library), and (b) splitting large objects over multiple rows of the library and using *multi-row queries* that retrieve (secretly) many rows in a single query. The reduced communication cost is close to the optimal: the size of the longest object in the library. However, this cost is still high, especially if a smaller object is being retrieved. An alternative is to download different rows (of an object) as independent objects, possibly at the cost of increasing the consumption delay [86]. Popcorn uses this technique for objects that are divided over multiple rows, but in addition reduces the number of such objects by using a combination of compression and padding (§2.3.4).

Prior PIR implementations. Many of the CPIR and ITPIR protocols described above have been implemented. The Percy++ library [116] contains several of them [36, 72, 88–90, 117, 134, 172]. Also, [133] is implemented as a fork of Percy++, RAID-PIR [86] is implemented on top of upPIR [67], and there are numerous CPIR implementations [34, 83, 101, 177, 181, 194, 210, 232, 242, 254], among which XPIR [34] is the fastest. Popcorn incorporates some of these implementations as modules: it uses the XPIR library for CPIR and borrows the CGKS ITPIR [72] code from Percy++. Sections 2.4 and 2.5 empirically or analytically compare Popcorn against these prior implementations.

2.7 Discussion, limitations, and future work

We evaluated Popcorn at the scale of a Netflix library, and found that the results are cautiously encouraging: compared to a baseline, I/O and CPU overhead are both lower (due to amortization, batching, and careful provisioning). And, although the overall resource cost is high, the *dollar* cost is manageable. Below, we discuss fundamental limitations of Popcorn, followed by limitations of the prototype and current design that require future work.

Fundamental limitations. We see three main limitations. First, because Popcorn's overheads grow linearly with the number of objects, it has no hope of scaling to YouTube-size libraries. Second, organizations that serve objects can collude to compromise Popcorn's privacy guarantee. Admittedly, an assumption of no collusion may be unrealistic against state-level adversaries that can compromise multiple organizations (or already have). Third, Popcorn cannot support forward seeks during playback: such user actions alter the download pattern in a content-dependent way, thus revealing information.

Library updates. To support online updates, Popcorn should execute both CPIR and ITPIR queries on the same version of the key and object libraries, at the key server and at both object servers. Standard solutions exist (e.g., generation numbers in concert with garbage collection).

Integration with CDNs. Running Popcorn on content delivery networks (CDNs) would present two main challenges: maintaining the utility of batching when running on a distributed infrastructure, and increased hardware provisioning at the CDN's edge servers. Though addressing the latter is non-trivial, we think that it does not require a paradigm shift: Akamai's EdgeComputing service [84] already enables running CPU-intensive enterprise business web applications at edge servers. Moreover, Netflix recently installed custom-built storage-optimized appliances at the edges.

Similarly, we think that, though the CDN's distributed infrastructure will reduce opportunities for batching, enough concurrency will remain to make the service cost effective. Indeed, rough back-of-the-envelope calculations suggest that request rates for Netflix are already quite high (e.g., over 9200 requests/90min/PoP[11]) and are growing fast [14]. This is not specific to Netflix: similar request rates (average of 6000 requests in 90 minutes from within a single city) have been reported for other video on demand systems [258].

Changes in load. Unless Popcorn is always wastefully provisioned for the peak load, load changes require care: the assignment of work units to machines depends on the number of clients (§2.3.3). A solution is to rely on virtual machines (VMs): give each VM a single slice, and then provide elasticity via VM migration or consolidation.

Variations in quality and bandwidth. *Adaptive streaming* lets clients switch between different video quality levels to adjust to bandwidth fluctuations. Popcorn could support this feature in two ways. First, it could maintain an individual library for each quality level. Clients would send queries to all libraries but download a video chunk only from the appropriate one. (A concern is, does switching between libraries leak information? No, because the chunk download pattern and switches are "lined up" with a reference object, O_{avg} (§2.3.4).) This solution is simple, but

[11] Assumes 10 billion hours watched in 3 months [13], requests are for a 90 minute video, and a total of 500 Points of Presence (PoP).

asking each library to process every request would increase server-side work significantly.

Alternatively, Popcorn could exploit layered coding [136, 152, 201, 213] or multiple description coding (MDC) [126, 204, 243]. There would be a single basic quality library accessed by all clients, with separate libraries for enhancement layers (better spatial resolution, bitrate, frame rate, etc.). The server-side work would thus be proportional only to the size of the highest quality library.

Billing and accounting. Popcorn must enable the content distributor to charge consumers, pay royalties, and collect aggregate statistics. The current prototype can support both subscription-based and pay-per-view pricing models, by monitoring accesses to the key server. Furthermore, by default it works with a prepaid royalty model, where the distributor pays a fixed license fee up front. However, in its current form, Popcorn does not support advanced pricing models (different prices for different objects, possibly in tiers) or advanced royalties models (e.g., based on number of views or aggregate statistics). However, we think that these limitations are not fundamental, as prior works [37, 65, 134, 229] have addressed them in different contexts. Future work is to investigate the performance and privacy implications of composing these works with Popcorn.

Targeted ads and recommendation services. Popcorn does not currently support targeted advertisements or recommendations. Incorporating relevant prior work [47, 63, 66, 129, 150, 157] into Popcorn is a direction for future work.

Chapter 3

Pretzel: A privacy-preserving email system

Email, in contrast to on-demand media, is ubiquitous and fundamental. For many, it is the principal communication medium, even with intimates. For these reasons, and others outlined in Chapter 1 (hacks of centralized servers, snooping by rogue employees, etc.), our animating ideal is that email should be *end-to-end private by default*.

How far are we from this ideal? On the plus side, *hop-by-hop* encryption has brought encouraging progress in protecting email privacy against a range of network-level attacks. Specifically, many emails now travel between servers over encrypted channels (TLS [92, 106]). And network connections between the user and the provider are often encrypted, for example using HTTPS (in the case of webmail providers) or VPNs (in the case of enterprise email accounts).

However, emails are not by default encrypted end-to-end between the two clients: intermediate hops, such as the sender's and receiver's email provider, handle emails in plaintext. A crucial reason emails are not encrypted end-to-end by default—at least the one that is often cited [75, 76, 105, 122, 223]—is that encryption appears to be incompatible with value-added functions (such as spam filtering), as noted earlier (Chapter 1). These functions are proprietary; for example, the provider might have invested in training a spam filtering model, and does not want to publicize it (even if a dedicated party can infer it [230]). So it follows that the functions must execute on providers' servers with access to plaintext emails.

But does that truly follow? In this chapter, we describe Pretzel, a private email system that refutes these claims on incompatibility. At a high-level, Pretzel relies on two cryptographic primitives: end-to-end encryption and secure two-party computation (2PC); 2PC protocols enable one or both parties to learn the output of an agreed-upon function (for example, spam filtering), without revealing the inputs (spam filter and email) to each other.

The challenge in Pretzel comes from the 2PC component. There is a tension between expressive power (the best 2PC schemes can handle any function and even hide it from one of the two parties) and cost (those schemes remain exorbitant, despite progress in lowering the costs; §3.2.2). Therefore, Pretzel makes certain compromises (§3.1.1) to gain even the possibility of plausible performance: baking in specific algorithms, requiring both the algorithms' logic and the model features to be exposed (model parameters are hidden), and incurring per-function design work.

Pretzel's central example is classification, which it applies to both spam filtering and topic extraction (the implementation also includes a simple keyword search function). Pretzel's first step is to compose (a) a relatively efficient 2PC protocol (§3.2.2) geared to computations that consist mostly of linear operations [33, 54, 142, 196, 208], (b) linear classifiers from machine learning (Naive Bayes, SVMs, logistic regression), which fit this form and which have good accuracy (§3.2.1), and (c) mechanisms that protect against adversarial parties. Although the precise protocol (§3.2.3) has not appeared before, we don't claim it as a contribution, as its elements are well-understood. This combination is simply the jumping-off point for Pretzel.

The work of Pretzel is adapting and incorporating this baseline into a system for end-to-end encrypted email. In this context, the costs of the baseline would be, if not quite outlandish, nevertheless too high. Pretzel responds with several protocol refinements: replacing the cryptosystem (§3.3.1), conserving calls into it by applying a packing technique [112] (§3.3.2), and decomposing classification into a non-private and a private step (§3.3.3). In addition, Pretzel applies well-known ideas (feature selection to reduce costs, various mechanisms to guard against misuses of the protocol).

None of the elements of Pretzel are individually remarkable. However, taken together, they produce the first (to our knowledge) demonstration that classification can be done privately, at tolerable cost, in the email setting (§3.5).

3.1 Architecture and design of Pretzel

3.1.1 Design ethos: (non)requirements

Pretzel would ideally (a) enable rich computation (of functions such as spam filtering, topic extraction, and predictive personal assistance) over email, (b) hide the inputs and implementations of those computations, and (c) impose little overhead. But these three ideals are in tension. Below we describe the compromises that form Pretzel's design ethos.

- *Functionality.* We will not insist that Pretzel replicate *exactly* the computations that providers such as Google perform over email; in fact, we don't actually know in detail what they do. Rather, we aim to approximate the value-added functions that they provide.
- *Provider privacy.* Related to the prior point, Pretzel will not support proprietary algorithms; instead, Pretzel will protect the *inputs* to the algorithms. For example, all users of Pretzel will know the spam filtering model (both its structure and its features), but the parameters to the model will be proprietary.
- *User privacy.* Pretzel will not try to enshroud users' email in complete secrecy; indeed, it seems unavoidable that computing over emails would reveal some information about them. However, Pretzel will be designed to reveal only the outputs of the computation, and these outputs will be short (in bits).
- *Threat model and maliciousness.* Pretzel will not build in protection against actions that subvert the protocol's *semantics* (for example, a provider who follows the protocol to the letter but who designs the topic extraction model to recover a precise email); we will deal with this issue by relying on context, a point we elaborate on later (§3.3.4, §3.6). Pretzel will, however, build in defenses against adversaries that deviate from the protocol's *mechanics*; these defenses will not assume particular misbehaviors, only that adversaries are subject to normal cryptographic hardness.
- *Performance and price.* Whereas the status quo imposes little overhead on email clients, Pretzel will incur network, storage, and computation overhead at clients. However, Pretzel will aim to limit the network overhead to small multiples of the overhead in the status quo, the storage cost to several hundred megabytes, and the CPU cost to a few hundred milliseconds of time per processed email. For

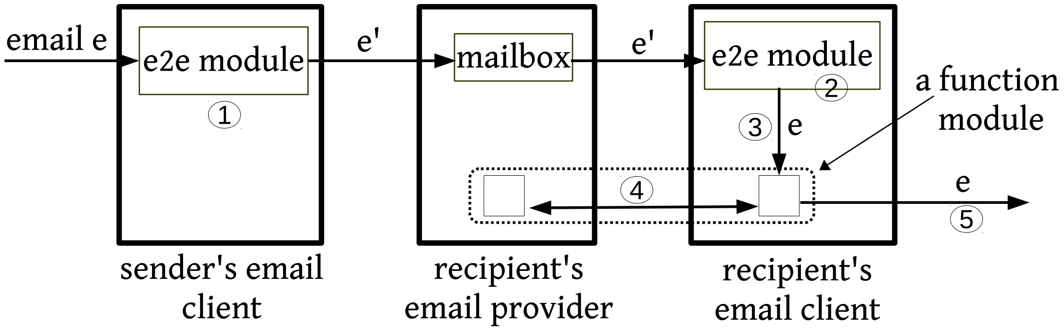

Figure 3.1: Pretzel's architecture. e denotes plaintext email; e' denotes encrypted email. The sender's provider is not depicted.

the provider, Pretzel's aim is to limit overheads to small multiples of the costs in the status quo.

- *Deployability and usability.* Certain computations, such as encryption, will have to run on the client. However, web applications are permitted to consume client-side resources, including storage [74]. Furthermore, Pretzel will also aim to be configuration-free. Finally, Pretzel must be backwards compatible with existing email delivery infrastructure (SMTP, IMAP, etc.).

3.1.2 Architecture

Figure 3.1 shows Pretzel's architecture. Pretzel comprises an *e2e module* and *function modules*. The e2e module implements an end-to-end encryption scheme; a function module implements a computation over the email content (spam filtering, etc.). The e2e module is client-side only, while a function module has a component at the client and another at the provider.

At a high level, Pretzel works as follows. An email sender uses its e2e module to encrypt and sign an email for an email recipient (step ①). The recipient uses its e2e module to authenticate and decrypt the email (step ②). The e2e module can implement any end-to-end encryption scheme; Pretzel's current prototype uses OpenPGP [16, 64]. Next, the recipient passes the decrypted email contents to the client-side components of the function modules (step ③), which then participate in a protocol with their counterparts at the provider (step ④). At the end of the protocol, either the client or the provider learns the output of the computation (for example, a bit encoding whether the email is spam or not). Finally, the client processes the decrypted email according to the output (for example, labels it as spam), and delivers it to the recipient (step ⑤).

Pretzel's e2e module requires cryptographic keys for encrypting, decrypting, signing, and verifying, and also demands a solution to manage these keys (for instance, share keys across devices) [8, 55, 180, 244]. However, the problem of managing keys is one of the obstacles (§3.6) that Pretzel does not address.

The main work for Pretzel surrounds the function modules; the challenge is to balance privacy, functionality, and performance (§3.1.1). Our focus will be on two modules: spam filtering and topic extraction (§3.2, §3.3). We will also report on an elementary keyword search module (§3.4). But before delving into details, we walk through some necessary background on the class of computations run by these modules and the cryptographic protocols that they build on.

3.2 Background, baseline, and related work

3.2.1 Classification

Spam filtering and topic extraction are classification problems and, as such, require *classifier algorithms*. Pretzel is geared to linear classifiers. So far, we have implemented Naive Bayes (NB) [127, 179, 183, 205] classifiers, specifically a variant of Graham-Robinson's NB [127, 205] for spam filtering (we call this variant GR-NB),[1] and multinomial NB [179] for topic extraction; Logistic Regression (LR) classifiers [108, 120, 167, 189], specifically binary LR [167] and multinomial LR [108] for spam filtering and topic extraction respectively; and linear Support Vector Machine (SVM) classifiers [56, 81, 151, 215], specifically two-class and one-versus-all SVM [56] for spam filtering and topic extraction respectively. These algorithms, or variants of them, yield high accuracy [79, 120, 127, 137, 151, 263] (see also §3.5.1, §3.5.2), and are used in popular open-source software packages for spam filtering, classification, and general machine learning [2, 21, 22, 27, 28, 108].

The three types of classifiers differ in their underlying assumptions and how they learn parameters from training data. However, when applying a trained

[1] The original Graham-Robinson NB protects against spam emails that hide a short message within a large non-spam text [128]. We do not implement that piece; the resulting change in classification accuracy is small (§3.5.1).

model, they all perform analogous linear operations. We will use Naive Bayes as a running example, because it is the simplest to explain.

Naive Bayes classifiers. These algorithms assume that a document can belong to one of several *categories* (for example, spam or non-spam). The algorithms output a prediction of a document's category.

Documents (emails) are represented by *feature vectors* $\vec{x} = (x_1, \ldots, x_N)$, where N is the total number of features. A feature can be a word, a group of words, or any other efficiently computable aspect of the document; the algorithms do not assume a particular mapping between documents and feature vectors, only that some mapping exists. In the GR-NB spam classifier [127, 205], x_i is Boolean, and indicates the presence or absence of feature i in the document; in the multinomial NB text classifier, x_i is the frequency of feature i.

The algorithms take as input a feature vector and a *model* that describes the categories. A model is a set of vectors $\{(\vec{v}_j,\, p(C_j))\}$ $(1 \leq j \leq B)$, where C_j is a category (for example, spam or non-spam), and B is the number of categories (two for spam; 2208 for topics, based on Google's public list of topics [123]). $p(C_j)$ denotes the assumed a priori category distribution. The ith entry of $\vec{v}_j$ is denoted $p(t_i \mid C_j)$ and is, roughly speaking, the probability that feature i, call it t_i, appears in documents whose category is C_j.[2]

The GR-NB spam classification algorithm labels an email, as represented by feature vector $\vec{x}$, as spam if $p(\text{spam} \mid \vec{x})$ is greater than some fixed threshold. To do so, the algorithm computes $\alpha = 1/p(\text{spam} \mid \vec{x}) - 1$ in log space. One can show (Appendix C.1) that $\log \alpha$ is equivalent to:

$$\begin{aligned} &\left(\sum_{i=1}^{i=N} x_i \cdot \log p(t_i \mid C_2)\right) + 1 \cdot \log p(C_2) \\ -&\left(\sum_{i=1}^{i=N} x_i \cdot \log p(t_i \mid C_1)\right) + 1 \cdot \log p(C_1), \end{aligned} \tag{3.1}$$

[2] In more detail, the GR-NB spam classifier assumes that the $\{x_i\}$ are realizations of independent, separate Bernoulli random variables (RVs), with the probabilities of each RV, $p(t_i \mid C_j)$, depending on the hypothesized category. The multinomial NB text classifier assumes that the $\{x_i\}$ follow a multinomial distribution, with N bins and $\sum_i x_i$ trials, where the bin probabilities are $p(t_i \mid C_j)$ and depend on the hypothesized category.

where C_1 represents spam and C_2 represents non-spam.

For the multinomial NB text classifier, selection works by identifying the category C_{j^*} that maximizes likelihood: $j^* = \operatorname{argmax}_j\, p(C_j \mid \vec{x})$. One can show (Appendix C.2) that it suffices to select the C_j for which the following is maximal:

$$\left(\sum_{i=1}^{i=N} x_i \cdot \log p(t_i \mid C_j)\right) + 1 \cdot \log p(C_j). \tag{3.2}$$

For LR and SVM classifiers, the term $\log p(t_i \mid C_j)$ is replaced by a "weight" term $w_{i,j}$ for feature x_i and category C_j, and $\log p(C_j)$ is replaced by a "bias" term b_j for category j.

3.2.2 Secure two-party computation

To perform the computation described above within a function module (§3.1.2) securely, that is, in a way that the client does not learn the model parameters and the provider does not learn the feature vector, Pretzel uses *secure two-party computation* (2PC): cryptographic protocols that enable two parties to compute a function without revealing their inputs to each other [118, 252]. Pretzel builds on a relatively efficient 2PC protocol [33, 54, 142, 196, 208] that we name **Yao+GLLM**; we present this below, informally and bottom up (for details and rigorous descriptions, see [115, 138, 169, 209]).

Yao's 2PC A building block of Yao+GLLM is the classic scheme of Yao [252]. Let f be a function, represented as a Boolean circuit (meaning a network of Boolean gates: AND, OR, etc.), with n-bit input, and let there be two parties P_1 and P_2 that supply separate pieces of this input, denoted x_1 and x_2, respectively. Then Yao (as the protocol is sometimes known), when run between P_1 and P_2, takes as inputs f and x_1 from P_1, x_2 from P_2, and outputs $f(x_1, x_2)$ to P_2, such that P_1 does not learn anything about x_2, and P_2 does not learn anything about x_1 except what can be inferred from $f(x_1, x_2)$.

At a very high level, Yao works by having one party generate encrypted truth tables, called *garbled Boolean gates*, for gates in the original circuit, and having the other party decrypt and thereby evaluate the garbled gates.

In principle, Yao handles arbitrary functions. In practice, however, the costs are high. A big problem is the computational model. For example, 32-bit multiplication, when represented as a Boolean circuit, requires on the order of 2,000 gates, and each of those gates induces cryptographic operations (encryption, etc.). Recent activity has improved the costs (see [124, 140, 158, 160, 168, 224, 260, 261] and references therein), but the bottom line is still too expensive to handle arbitrary computations. Indeed, Pretzel's prototype uses Yao very selectively—just to compute several comparisons of 32-bit numbers—and even then it turns out to be a bottleneck (§3.5.1, §3.5.2), despite using a recent and optimized implementation [259, 260].

Secure dot products. Another building block of Yao+GLLM is a secure dot product protocol, specifically GLLM [115]. Many such protocols (also called secure scalar product (SSP) protocols) have been proposed [39, 46, 100, 102–104, 115, 146, 218, 231, 234, 249, 265]. They fall into two categories: those that are provably secure [100, 115, 249] and those that either have no security proof or require trusting a third party [39, 46, 102–104, 146, 218, 231, 234, 265]. Several protocols in the latter category have been attacked [70, 115, 141, 156, 164]. GLLM [115] is in the first category, is state of the art, and is widely used.

Hybrid: Yao+GLLM. Pretzel's starting point is Yao+GLLM, a hybrid of Yao and GLLM. It is depicted in Figure 3.2. One party starts with a matrix, and encrypts the entries. The other party starts with a vector and leverages additive (not fully) homomorphic encryption (AHE) to (a) compute the vector-matrix product in cipherspace, and (b) blind the resulting vector. The first party then decrypts to obtain the blinded vector. The vector then feeds into Yao: the two parties remove the blinding and perform some computation ϕ.

Yao+GLLM has been applied to spam filtering using LR [196], face recognition using SVM [33], and face and biometric identification using Euclidean distance [54, 142, 208].

Other related work. There are many works on private classification that do not build on Yao+GLLM. They rely on alternate building blocks or hybrids: additively homomorphic encryption [57, 170], fully homomorphic encryption [159] (FHE), or a different Yao hybrid [62]. For us, Yao+GLLM appeared to be a more promising starting point. For example, in contrast to the protocol of Khedr et al. [159], Yao+GLLM reveals only the final output of the computation rather than intermediate dot products. As another example, the resource consumption in Yao+GLLM is considerably lower than in Bost et al. [57].[3]

Another related line of research focuses on privacy and linear classifiers—but in the training phase. Multiple parties can train a global model without revealing their private inputs [235, 237, 251, 255–257], or a party can release a trained "noisy" model that hides its training data [155, 236, 264]. These works are complementary to Pretzel's focus on applying trained models.

3.2.3 Baseline protocol

Pretzel begins by applying the Yao+GLLM protocol (Figure 3.2, §3.2.2) to the algorithms described in Section 3.2.1. This works because expressions (3.1) and (3.2) are dot products of the necessary form. Specifically, the provider is party X and supplies $(\vec{v}_j,\, p(C_j))$; the client is party Y and supplies $(\vec{x},\, 1)$, which it obtains from an email using a feature extraction algorithm supplied by the provider (§3.1.1); and the protocol computes their dot product. Then, the threshold comparison (for spam filtering) or the maximal selection (for topic extraction) happens inside an instance of Yao. For spam filtering, the client receives the classification output; for topic extraction, the provider does. Note that storing the encrypted model at the client is justified by an assumption that model vectors change infrequently [78, 214, 215].

In defining this baseline, we include mechanisms to defend against adversarial parties (§3.1.1). Specifically, whereas under the classical Yao protocol an actively adversarial party can obtain the other's private inputs [147], Pretzel incorporates a variant [147, 158] that solves this problem. This variant brings some additional expense, but that expense can be incurred during the setup phase and amortized. Also, Yao+GLLM assumes that the AHE's key generation is done hon-

[3] For the data point at $N = 70$, $B = 24$ (these variables are defined in Figure 3.2), Bost et al. report network transfers and computation times (for the two parties) of 1911 KB, 1664 ms, and 1652 ms [57], whereas these overheads are 156.1 KB, 757.9 ms, and 8.6 ms for our implementation of Yao+GLLM (§3.4) on comparable hardware. These differences in overheads are due to a packing optimization in Yao+GLLM and improvements (see the pointers to recent activity above) that reduce the overheads of Yao.

Yao+GLLM

- The protocol has two parties. Party X begins with a matrix; Party Y begins with a vector. The protocol computes a vector-matrix product and then performs an arbitrary computation, ϕ, on the resulting vector; neither party's input is revealed to the other.
- The protocol assumes an additively homomorphic encryption (AHE) scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, meaning that $\mathsf{Enc}(pk, m_1) \cdot \mathsf{Enc}(pk, m_2) = \mathsf{Enc}(pk, m_1 + m_2)$, where m_1, m_2 are plaintext messages, $+$ represents addition of two plaintext messages, and $\cdot$ is an operation on the ciphertexts. This also implies that given a constant z and $\mathsf{Enc}(pk, m_1)$, one can compute $\mathsf{Enc}(pk, z \cdot m_1)$.

Setup phase

1. Party X forms a matrix with columns $\vec{v}_1, \ldots, \vec{v}_B$; each vector has N components. It does the following:
   (a) Generates public and secret keys $(pk, sk) \leftarrow \mathsf{Gen}(1^k)$, where k is a security parameter.
   (b) Using an AHE scheme, encrypts each column of the matrix component-wise. That is, computes $\mathsf{Enc}(pk, \vec{v}_j) = (\mathsf{Enc}(pk, v_{1,j}), \ldots, \mathsf{Enc}(pk, v_{N,j}))$.
   (c) Sends the encrypted matrix columns and pk to Party Y.

Computation phase

2. Party Y begins with an N-component vector $\vec{x} = (x_1, \ldots, x_N)$. It does the following:
   (a) (dot products) Computes encrypted dot product for each matrix column: $\mathsf{Enc}(pk, d_j) = \mathsf{Enc}(pk, \sum_{i=1}^{N} x_i \cdot v_{i,j})$, this abuses notation, since the encryption function is not deterministic. The computation relies on the homomorphic property.
   (b) (blinding) Blinds d_j by adding random noise $n_j \in_R \{0,1\}^{b+\delta}$. That is, computes $\mathsf{Enc}(pk, d_j + n_j) = \mathsf{Enc}(pk, d_j) \cdot \mathsf{Enc}(pk, n_j)$. Here b is the bit-length of d_j and $\delta \geq 1$ is a security parameter.
   (c) Sends $(\mathsf{Enc}(pk, d_1 + n_1), \ldots, \mathsf{Enc}(pk, d_B + n_B))$ to Party X.
3. Party X applies Dec component-wise, to get $(d_1 + n_1, \ldots, d_B + n_B)$
4. Party X and Party Y participate in Yao's 2PC protocol; they use a function f that subtracts the noise n_j from $d_j + n_j$ and applies the function ϕ to the d_j. One of the two parties (which one depends on the arrangement) obtains the output $\phi(d_1, \ldots, d_B)$.

Figure 3.2: Yao+GLLM. This protocol [33, 54, 142, 196, 208] combines GLLM's secure dot products [115] with Yao's general-purpose 2PC [252]. Pretzel's design and implementation apply this protocol to the linear classifiers described in §3.2.1. The provider is Party X, and the client is Party Y. Pretzel's instantiation of this protocol incorporates several additional elements (§3.2.3): a variant of Yao [147, 158] that defends against actively adversarial parties; amortization of the expense of this variant via precomputation in the setup phase; a technique to defend against adversarial key generation (for example, not invoking Gen correctly); and a packing technique (§3.3.2) in steps 1b and 2a.

estly, whereas we would prefer not to make that assumption; Pretzel incorporates the standard response.[4]

While the overall baseline is literally new (Yao+GLLM was previously used in weaker threat models, etc.), its elements are well-known, so we do not claim novelty.

3.3 Pretzel's protocol refinements

The baseline just described is a promising foundation for private classification. But adapting it to an end-to-end system for encrypted email requires work. The main issue is costs. As examples, for a spam classification model with $N = 5\text{M}$ features, the protocol consumes over 1 GB of client-side storage space; for topic extraction with $B = 2048$ categories, it consumes over 150 ms of provider-side CPU time and 8 MB in network transfers (§3.5). Another issue to consider is the robustness of the guarantees.

This section describes Pretzel's refinements, adjustments, and modifications. The nature of the work varies from low-level cryptographic optimizations, to architectural rearrangement, to applications of known ideas (in which case the work is demonstrating that they are suitable here). We begin with refinements that are aimed at reducing costs (§3.3.1–§3.3.3), the effects of which are summarized in Figure 3.3; then we describe Pretzel's robustness to misbehaving parties (§3.3.4).

3.3.1 Replacing the cryptosystem

Both Pretzel and the baseline require additively homomorphic encryption (Figure 3.2). The traditional choice for AHE—it is used in prior works [33, 142, 196, 208]—is Paillier [193], which is based on a longstanding number-theoretic presumed hardness assumption. However, Paillier's Dec takes hundreds of microseconds on a modern CPU, which contributes substantially to provider-side CPU time.

Instead, Pretzel turns to a cryptosystem based on the *Ring-LWE assumption* [173], a relatively young assumption (which is usually a disadvantage in cryptography) but one that has nonetheless received a lot of recent attention [31, 59, 85,

[4] In more detail, the AHE has public parameters which, if chosen adversely (non-randomly) would undermine the expected usage. To get around this, Pretzel determines these parameters with Diffie-Hellman key exchange so that both parties inject randomness into these parameters [93, 98, 182, 216].

	Non-private	Baseline (§3.2.3)	Pretzel (§3.3.1–§3.3.3)
Setup			
Provider CPU time	N/A	$N \cdot \beta_{\text{pail}} \cdot e_{\text{pail}} + K_{\text{cpu}}$	$N' \cdot \beta'_{\text{xpir}} \cdot e_{\text{xpir}} + K_{\text{cpu}}$
Client CPU time	N/A	K_{cpu}	K_{cpu}
Network transfers	N/A	$N \cdot \beta_{\text{pail}} \cdot c_{\text{pail}} + K_{\text{net}}$	$N' \cdot \beta'_{\text{xpir}} \cdot c_{\text{xpir}} + K_{\text{net}}$
Client storage	N/A	$N \cdot \beta_{\text{pail}} \cdot c_{\text{pail}}$	$N' \cdot \beta'_{\text{xpir}} \cdot c_{\text{xpir}}$
Per-email			
Provider CPU	$L \cdot h + L \cdot B \cdot g$	$\beta_{\text{pail}} \cdot d_{\text{pail}} + B \cdot y_{\text{per-in}}$	$\beta''_{\text{xpir}} \cdot d_{\text{xpir}} + B' \cdot y_{\text{per-in}}$
Client CPU	N/A	$L \cdot \beta_{\text{pail}} \cdot a_{\text{pail}} + \beta_{\text{pail}} \cdot e_{\text{pail}} + B \cdot y_{\text{per-in}}$	$L \cdot \beta_{\text{xpir}} \cdot a_{\text{xpir}} + (L + B') \cdot s + \beta''_{\text{xpir}} \cdot e_{\text{xpir}} + B' \cdot y_{\text{per-in}}$
Network	sz_{email}	$sz_{\text{email}} + \beta_{\text{pail}} \cdot c_{\text{pail}} + B \cdot sz_{\text{per-in}}$	$sz_{\text{email}} + \beta''_{\text{xpir}} \cdot c_{\text{xpir}} + B' \cdot sz_{\text{per-in}}$

L = number of features in an email (§3.2.3)
B = number of categories in the model (§3.2.3)
sz_{email} = size of an email
$\beta_{\text{pail}} := \lceil B / p_{\text{pail}} \rceil, \beta_{\text{xpir}} := \lceil B / p_{\text{xpir}} \rceil$
e = encryption CPU time in an AHE scheme
d = decryption CPU time in an AHE scheme
a = homomorphic addition CPU time in an AHE scheme (Fig. 3.2)
$b = \log L + b_{\text{in}} + f_{\text{in}}$ (§3.3.2)
f_{in} = # of bits for the frequency of a feature in an email (§3.3.2)
$\beta'_{\text{xpir}} := \lfloor B / p_{\text{xpir}} \rfloor + 1 / \lfloor p_{\text{xpir}} / k \rfloor$ where $k = (B \bmod p_{\text{xpir}})$
$\beta''_{\text{xpir}} := \beta_{\text{xpir}}$ (if spam) or B' (if topics)

h = CPU time to extract a feature and lookup its conditional probabilities
g = CPU time to add two probabilities
N = number of features in the model (§3.2.3)
p = # of b-bit probabilities packed in a ciphertext (§3.3.2)
$K_{\text{cpu}}, K_{\text{net}}$ = constants for CPU and network costs (§3.2.3)
c = ciphertext size in an AHE scheme
$y_{\text{per-in}}, sz_{\text{per-in}}$ = Yao CPU time and network transfers per b-bit input (§3.2.2)
b_{in} = # of bits to encode a model parameter (§3.3.2)
N' = # of features selected after aggressive feature selection (§3.3.3) ($N' = N$ if spam)
B' = # of candidate topics ($\ll B$) (§3.3.3) ($B' = B$ if spam)
s = "left-shift" CPU time in XPIR-BV (§3.3.2)

Figure 3.3: Cost estimates for classification. Non-private refers to a system in which a provider locally classifies plaintext email. The baseline is described in Section 3.2.3. Microbenchmarks are given in §3.5.

174, 197, 206]. Specifically, Pretzel incorporates the additively homomorphic cryptosystem of Brakerski and Vaikuntanathan [59], as implemented and optimized by Melchor et al. [35] in the XPIR system; we call this XPIR-BV. This change brings the cost of each invocation of Dec down by over an order of magnitude, to scores of microseconds (§3.5), and similarly with Enc. The gain is reflected in the cost model (Figure 3.3), in replacing d_{pail} with d_{xpir} (likewise with e_{pail} and e_{xpir}, etc.)

However, the change makes ciphertexts 64× larger: from 256 bytes to 16 KB. Yet, this is not the disaster that it seems. Network costs do increase (in Figure 3.2, step 2c), but by far less than 64×. Because the domain of the encryption function (that is, the size of the plaintext space) grows, one can tame what would otherwise be a large increase in network and storage, and also gain further CPU savings. We describe this next.

3.3.2 Packing in Pretzel

The basic idea is to represent multiple plaintext elements (for example, model parameters) in a single ciphertext; this opportunity exists because the domain of Enc is much larger than any single element that needs to be encrypted. Using packing, one can reduce the number of invocations of Enc and Dec in Figure 3.2, specifically in step 1b, step 2b, and step 3. The consequence is a significant reduction in resource consumption, specifically client storage for spam filtering, and provider CPU time for topic extraction.

A common packing technique—it is used in GLLM [115], Pretzel's baseline (§3.2.3), and the works that build on GLLM [33, 142]—traverses each row in the matrix from left to right and encrypts together sets of elements, while restricting the packing to be within the given rows. Although better than no packing, this technique does not always fully utilize the space in a ciphertext. For example, when the number of elements in a matrix row is two (as in the spam filtering application) and the number of elements that can be packed together is 1024 (as in the XPIR-BV ciphertexts), then 1022 "slots" remain unutilized.

Recent packing techniques, proposed in the context of aggregation queries on encrypted databases [233] and homomorphic evaluation of AES-128 encryption [112], address the limitation described above, by packing across both columns and rows. These techniques traverse the matrix in row-major order without restrict-

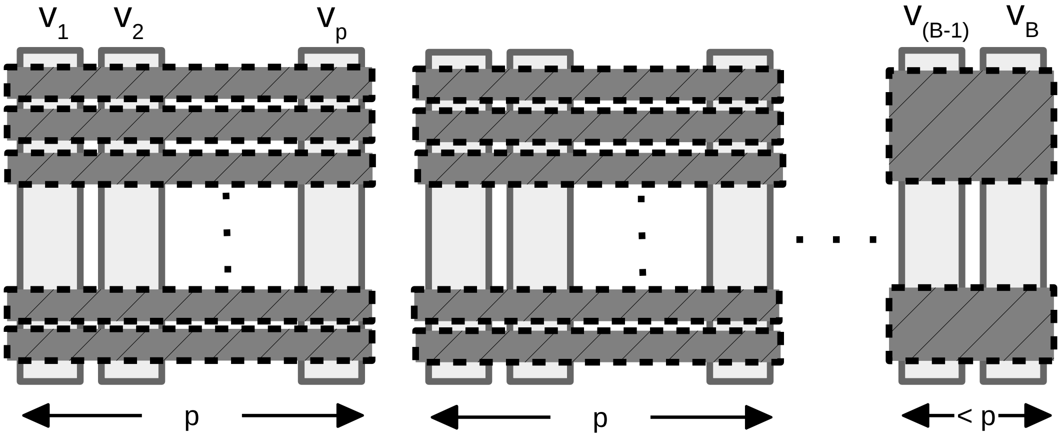

Figure 3.4: Packing in Pretzel. Light gray rectangles represent matrix columns $(\vec{v}_1, \ldots, \vec{v}_B)$; dark gray represent ciphertexts. The arrangement in matrices with p columns follows GLLM [115]; the matrix with $< p$ columns follows Gentry et al. [112].

ing the packing to be within a row (see the rightmost matrix in Figure 3.4), thereby utilizing the "empty slots" in a ciphertext.

Pretzel incorporates both types of techniques described above. Below, we describe the relevant details on how and where these techniques are incorporated.

Details. Let p be the number of elements that can be packed together in a ciphertext, and let b be the number of semantically useful bits in a dot product output. Then, in step 1b in Figure 3.2, Pretzel splits (not depicted in the figure) the matrix $\{(\vec{v}_j,\, p(C_j))\}$ into zero or more sets of p column vectors plus up to one set with fewer than p vectors as depicted in Figure 3.4. For the sets with p vectors, it packs together all p elements of a row [115]. For the last set, it packs elements in row-major order under one constraint: elements in the same row of the matrix must not be put into different ciphertexts [112, 233].

Then, to compute dot products (in step 2a in Figure 3.2) for all columns except those in the rightmost matrix (Figure 3.4), Pretzel uses the fact that that the elements that need to be added are aligned [115]. For example, if the elements in the first row $(v_{1,1}, \ldots, v_{1,p})$ are to be added to those in the second row $(v_{2,1}, \ldots, v_{2,p})$, then the ciphertext space operation applied to $c_1 = \mathsf{Enc}(pk, v_{1,1} \| \ldots \| v_{1,p})$ and $c_2 = \mathsf{Enc}(pk, v_{2,1} \| \ldots \| v_{2,p})$ yields $c_3 = c_1 \cdot c_2 = \mathsf{Enc}(pk, v_{1,1} + v_{2,1} \| \ldots \| v_{1,p} + v_{2,p})$. For this to work, the individual sums (for example, $v_{1,p} + v_{2,p}$) cannot overflow b bits.

For the columns in the rightmost matrix (Figure 3.4), Pretzel performs dot products by exploiting the homomorphism to cyclically rotate the packed elements in a ciphertext [112]. For example, assume $c = \mathsf{Enc}(pk,\ v_{1,1} \| \ldots \| v_{1,k} \| v_{2,1} \| \ldots \| v_{2,k})$ is a packed ciphertext, where $v_{1,1}, \ldots, v_{1,k}$ are elements from the first row, and $v_{2,1}, \ldots, v_{2,k}$ are from the second row. To add each $v_{1,i}$ with $v_{2,i}$ for $i \in \{1, \ldots, k\}$, one can left-shift elements in c by k positions to get $c' = \mathsf{Enc}(pk, v_{2,1} \| \ldots \| v_{2,k} \| \ldots)$; this is done by applying the "constant multiplication" operation (Figure 3.2, bullet 2), with $z = 2^{k \cdot b}$. At this point, the rows are lined up, and one can operate on c and c' to add the plaintext elements.

We haven't yet said how the values of p and b are determined. Let G denote the number of bits in the domain of the encryption algorithm Enc, b_{in} denote the number of bits required to represent an element that would be encrypted (a model parameter in our case), and f_{in} denote the number of bits for the multiplier of an encrypted element (frequency of a feature extracted from an email in our case). Then, the output of a dot product computation—assuming a sum of L products, each formed from a b_{in}-bit element and a f_{in}-bit element—has $b = \log L + b_{\text{in}} + f_{\text{in}}$ bits (in our context, L would be the number of features extracted from an email). This means that there is "room" to pack $p = \lfloor G/b \rfloor$ elements into a single ciphertext.

Cost savings. Here we give rough estimates of the effect of the refinements in this subsection and the previous; a more detailed evaluation is in Section 3.5. For the spam filtering module, the provider's CPU drops by $5\times$ and the client-side storage drops by $7\times$, relative to the baseline (§3.2.3). However, CPU at the client increases by $10\times$ (owing to the cyclic shifts), and the network overhead increases by $5.4\times$; despite these increases, both costs are not exorbitant in absolute terms, and we view them as tolerable (§3.5.1, §3.5.2). The provider-side costs for spam filtering are comparable to an arrangement where the provider classifies plaintext emails non-privately.

For the topic extraction module, the cost improvements relative to the baseline (§3.2.3) are smaller: provider CPU drops by $1.37\times$, client CPU drops by $3.25\times$, storage goes up by a factor of 2, and the network cost goes up slightly. Beyond that, the *non*-private version of this function is vastly cheaper than for spam, to the point that the private version is (depending on the resource) up to two orders of magnitude worse than the non-private version. The next subsection addresses this.

3.3.3 Pruning in topic extraction

Decomposed classification. So far, many of the costs are proportional to B: CPU and network cost of Yao (Figure 3.2, step 4), and storage (Figure 3.2, "setup phase"). For spam filtering, this is not a problem ($B = 2$) but for topic extraction, B can be in the thousands.

Pretzel's response is a technique that we call *decomposed classification*. To explain the idea, we regard topic extraction as abstractly mapping an email, together with a set S of cardinality B (all possible topics), down to a set S^* of cardinality 1 (the chosen topic), using a model with proprietary parameters. Pretzel decomposes this map into two:

(i) Map the email, together with the set S, to a set S' of cardinality B' (for example, $B' = 20$); S' comprises *candidate topics*. The client does this by itself.

(ii) Map the email, together with S', down to a set S'' of cardinality 1; ideally S'' is the same as S^* (otherwise, accuracy is sacrificed). This step relies on a proprietary model and is done using secure two-party machinery. Thus, the costs of the expensive part of the protocol are now proportional to B' rather than to B (the gain is reflected in Figure 3.3, the last two columns of the "per-email" rows).

For this arrangement to make sense, several requirements must be met. First, the client needs to be able to perform the map in step (i) locally. Here, Pretzel exploits an observation: topic lists (the set S) are public today [123]. They have to be, so that advertisers can target and users can set interests. Thus, a client can in principle use some *non*-proprietary classifier for step (i). Pretzel is agnostic about the source of this classifier; it could be supplied by the client, the provider, or a third party.[5]

Second, the arrangement needs to be accurate, which it is when S' contains S^*. Pretzel observes that although the classifier used in step (i) would not be honed, it doesn't need to be, because it is performing a far coarser task than choosing a single topic. Thus, in principle, the step (i) map might reliably produce accurate outputs—meaning that the true topic, S^*, is among the B' candidates—without much training, expertise, or other proprietary input. Our experiments confirm that indeed the loss of end-to-end accuracy is small (§3.5.2).

Finally, step (ii) must not reveal S' to the provider, since that would be more

[5] In our prototype, the provider supplies the classifier after training it on a very small fraction of

Pretzel's protocol for proprietary topic extraction, based on candidate topics

- The protocol has two parties. Party X begins with a matrix $\vec{v}_1, \ldots, \vec{v}_B$. Party Y begins with a vector $\vec{x} = (x_1, \ldots, x_N)$ and a list S' of $B' < B$ column indexes, where each index is between 1 and B; S' indicates a subset of the columns of matrix $\vec{v}$. The protocol constructs a vector from the product of $\vec{x}$ and the submatrix of $\vec{v}$ given by S', and outputs the column index (in $\vec{v}$) that corresponds to the maximum element in the vector-submatrix product; neither party's input is revealed to the other.
- The protocol has two phases: setup and computation. The setup phase is as described in Figure 3.2 but with the addition of packing from §3.3.2.

Computation phase

3. Party Y does the following:
   (a) (compute dot products) As described in Figure 3.2, step 2a, and §3.3.2. At the end of the dot product computations, it gets a vector of packed ciphertexts $\vec{pcts} = (\mathsf{Enc}(pk, d_1 \| \ldots \| d_p), \ldots, \mathsf{Enc}(pk, \ldots \| d_B \| \ldots))$, where d_i is the dot product of $\vec{x}$ and the i-th matrix column $\vec{v}_i$, and p is the number of b-bit positions in a packed ciphertext (§3.3.2).

   (b) (separate out dot products for the columns in S' from the rest) For each entry in S', i.e., $S'[j]$, makes a copy of the packed ciphertext containing $d_{S'[j]}$, and shifts $d_{S'[j]}$ to the left-most b-bit position in that ciphertext. Because each ciphertext holds p elements, the separation works by using the quotient and remainder of $S'[j]$, when divided by p, to identify, respectively, the relevant packed ciphertext and position within it. That is, for $1 \leq j \leq B'$, computes ciphertext $\mathsf{Enc}(pk, d_{S'[j]} \| \ldots) = \vec{pcts}[Q_j] \cdot 2^{b \cdot R_j}$, where $Q_j = \lceil S'[j]/p \rceil - 1$, and $R_j = (S'[j] - 1) \bmod p$. The shifting relies on the multiply-by-constant homomorphic operation (see Figure 3.2 and §3.3.2).

   (c) (blinding) Blinds $d_{S'[j]}$ using the technique described in Figure 3.2, step 2b, but extended to packed ciphertexts. Sends the B' ciphertexts $(\mathsf{Enc}(pk, d_{S'[1]} + n_1 \| \ldots), \ldots, \mathsf{Enc}(pk, d_{S'[B']} + n_{B'} \| \ldots))$ to Party X. Here, n_j is the added noise.
4. Party X applies Dec on the B' ciphertexts, followed by bitwise right shift on the resulting plaintexts, to get $d_{S'[1]} + n_1, \ldots, d_{S'[B']} + n_{B'}$.
5. The two parties engage in Yao's 2PC. Party Y supplies S' and $\{n_j\}$ for $1 \leq j \leq B'$; Party X supplies $\{(d_{S'[j]} + n_j)\}$ for $1 \leq j \leq B'$; and, the parties use a function f that subtracts n_j from $d_{S'[j]} + n_j$, and computes and returns $S'[\text{argmax}_j \, d_{S'[j]}]$ to Party X.

Figure 3.5: Protocol for proprietary topic extraction, based on candidate topics (this instantiates step (ii) in Section 3.3.3). The provider is Party X; the client is Party Y. This protocol builds on the protocol presented in §3.2.3–§3.3.2.

information than a single extracted topic. This rules out instantiating step (ii) by naively applying the existing protocol (§3.2.3–§3.3.2), with S' in place of S. Pretzel's response is depicted in Figure 3.5. There are some low-level details to handle because of the interaction with packing (§3.3.2), but at a high level, this protocol works as follows. The provider supplies the entire proprietary model (with all B topics); the client obtains B dot products, in encrypted form, via the inexpensive component of Yao+GLLM (secure dot product). The client then extracts and blinds the B' dot products that correspond to the candidate topics. The parties finish by using Yao to privately identify the topic that produced the maximum.

Feature selection. Protocol storage is proportional to N (Figure 3.2, "setup phase"). Pretzel's response is the standard technique of *feature selection* [228]: incorporating into the model the features most helpful for discrimination. This takes place in the "setup phase" of the protocol (the number of rows in the provider's matrix reduces from N to N'; for the resulting cost reductions, see the last two columns of the "setup" rows in Figure 3.3). Of course, one presumes that providers already prune their models; the proposal here is to do so more aggressively. Section 3.5.2 shows that in return for large drops in the number of considered features, the accuracy drops only modestly. In fact, reductions of 75% in the number of features is a plausible operating point.

Cost savings. Feature selection reduces client-storage costs by a factor of N/N'. For $B = 2048$, $B' = 20$, and $L = 692$ (average number of features per email in the authors' emails), relative to the protocol in §3.3.2, the provider CPU drops by $45\times$, client CPU drops by $8.4\times$, and the network transfers drop by $20.4\times$ (§3.5.2). Thus, the aforementioned two orders of magnitude (above the non-private version) becomes roughly $5\times$.

3.3.4 Robustness to misbehaving parties

Pretzel aims to provide the following guarantees, even when parties deviate from the protocol:

its dataset.

1. The client and provider cannot (directly) observe each other's inputs nor any intermediate state in the computation.
2. The client learns at most 1 bit of output each time spam classification is invoked.
3. The provider learns at most $\log B$ bits of output per email. This comes from topic extraction.

Guarantee (1) follows from the baseline protocol, which includes mechanisms that thwart the attempted subversion of the protocol (§3.2.3). Guarantee (2) follows from Guarantee (1) and the fact that the client is the party who gets the spam classification output. Guarantee (3) follows similarly, provided that the client feeds each email into the protocol at most once; we discuss this requirement shortly.

Before continuing, we note that the two applications are asymmetric. In spam classification, the client, who gets the output, could conceivably try to learn the provider's model; however, the provider does not directly learn anything about the client's email. With topic extraction, the roles are reversed. Because the output is obtained by the provider, what is potentially at risk is the privacy of the email of the client, who instead has no access to the provider's model.

Leakage. Despite its guarantees about the number of output bits, Pretzel has nothing to say about the meaning of those bits. For example, in topic extraction, an adversarial provider could construct a tailored "model" to attack an email (or the emails of a particular user), in which case the $\log B$ bits could yield important information about the email. A client who is concerned about this issue has several options, including opting out of topic extraction (and presumably compensating the provider for service, since a key purpose of topic extraction is ad display, which generates revenue). We describe a more mischievous response below (in "Integrity").

In the spam application, an adversarial client could construct emails to try to infer model parameters, and then leak the model. Such leakage would not only undermine the proprietary nature of the model but also make it easier for spammers to bypass the spam filter [53, 241]. A possible defense would be for the provider to periodically revise the model (and maintain different versions).

Repetition and replay. An adversarial provider could conceivably replay a given email to a client k different times, each time with a unique topic model. The provider would then get $k \log B$ bits from the email, rather than $\log B$. Our defense is simply for the client to regard email transmission from each sender's device as a separate asynchronous—and lossy and duplicating—transmission channel. Solutions to detecting duplicates over such channels are well-understood: counters, windows, etc. Something to note is that, for this defense to work, emails have to be signed, otherwise an adversary can deny service by pretending to be a sender and spuriously exhausting counters.

Integrity. Pretzel does not offer any guarantees about which function Yao actually computes. For topic extraction, the client could, rather than garbling argmax (§3.2.2), instead garble an arbitrary function. Similarly, a client could input bogus candidate topics in step (ii) of decomposed classification (§3.3.3). In such cases, the aforementioned guarantees continue to hold (no inputs are disclosed, etc.), though of course this misbehavior interferes with the ultimate functionality. Pretzel does not defend against this case, and in fact, it could be considered a feature—it gives the client a passive way to "opt out", with plausible deniability (for example, the client could garble a function that produces an arbitrary choice of index).

The analogous attack, for spam, is for the provider to garble a function other than threshold comparison. This would undermine the spam/nospam classification and would presumably be disincentivized by the same forces incentivizing providers to supply spam filtering as a service in the first place.

3.4 Implementation

Our prototype fully implements the design described in Section 3.3. In addition, it includes an elementary keyword search module in which the client maintains and queries a client-side search index. The modules, written in 5,300 lines of C++ and 160 lines of Python, glue the code we borrow from existing libraries: GPGME [6] for OpenPGP encryption, Obliv-C [259] for Yao's 2PC protocol,[6] XPIR [35] for the

[6] Another choice would have been TinyGarble [224]. We found the performance of Obliv-C and TinyGarble to be comparable for the functions we compute inside Yao in Pretzel; we choose the former because it is easier to integrate with Pretzel's C++ code.

XPIR-BV AHE scheme, LIBLINEAR [45, 108] to train LR and SVM classifiers, and SQLite FTS4 [23] for the search index.

3.5 Evaluation

Our evaluation answers the following questions:

1. What are the provider- and client-side overheads of Pretzel? For what configurations (model size, email size, etc.) are they low?
2. How much do Pretzel's optimizations (§3.3) help in reducing the overheads?
3. How accurate are Pretzel's functions (for example, how accurately can they filter spam emails or extract topics of emails)?

A summary of evaluation results is as follows:

- Pretzel's provider-side CPU consumption for spam filtering and topic extraction is, respectively, 0.65 and 1.03–1.78× of a non-private arrangement, and, respectively, 0.17× and 0.01–0.02× of its baseline (§3.2.3). (One of the reasons that provider-side CPU consumption is low—and sometimes lower than in a non-private arrangement—is that the protocols shift work to the client.)
- Network transfers in Pretzel are 2.7–5.4× of a non-private arrangement, and 0.024–0.048× of its baseline (§3.2.3).
- Pretzel's client-side CPU consumption is less than 1s per email, and storage space use is a few hundred MBs. These are a few factors lower than in the baseline (§3.2.3).
- For topic extraction, the potential coarsening effects of Pretzel's classifiers (§3.3.3) are a drop in accuracy of between 1–3%.

Method and setup. We consider spam filtering, topic extraction, and keyword search separately.

For spam filtering and topic extraction, we compare Pretzel to its starting baseline, which we call **Hybrid** (this baseline is described in detail in Section 3.2.3 and Figure 3.2), and **NoPriv**, which models the status quo, in which the provider locally runs classification on plaintext email contents. For the keyword search function, we consider only the basic client-side search index based scheme (§3.4).

We vary the following parameters: number of features (N) and categories (B) in the classification models, number of features in an email (L), and the number of candidate topics (B') in topic extraction. For the classification models, we use synthetic datasets for measuring resource overheads, and real-world datasets for measuring accuracies. To generate synthetic emails, we use random words (between 4 to 12 letters each), and consider each word as one feature. For real-world data, we use the Ling-spam [40] (481 spam and $2{,}411$ non-spam emails), Enron [4] ($17{,}148$ spam and $16{,}555$ non-spam emails of about 150 Enron employees), and Gmail (355 spam and 600 non-spam emails received by one of the authors over a period of one month) datasets for spam filtering evaluation, and the 20 Newsgroup [7] ($18{,}846$ Usenet posts on 20 topics), Reuters-21578 [20] ($12{,}603$ newswire stories on 90 topics), and RCV1 [166] ($806{,}778$ newswire stories from 296 regions) datasets for topic extraction evaluation. To extract features from the documents in real-world datasets, we use the feature extraction algorithms from SpamBayes [22] and scikit-learn [21].

We measure resource overheads in terms of provider- and client-side CPU times to process an email, network transfers between provider and client, and the storage space used at a client. The resource overheads are independent of the classification algorithm (NB, LR, SVM), so we present them once; the accuracies depend on the classification algorithm, so we present them for each algorithm. To measure accuracies for spam classification, we use 10-fold cross validation experiments [61]; for topic extraction, we train a model on the training part of the datasets, and then apply it to the documents in the testing part.

Our testbed is Amazon EC2. We use a single `m3.2xlarge` machine for the provider and one machine of the same type for a client. At the provider, we use an independent CPU for each function module (§3.1.2). Similarly, the client uses a single CPU.

Microbenchmarks. Figure 3.6 shows the CPU and network costs for the common operations (Figure 3.3) in Pretzel and the baselines. We will use these microbenchmarks to explain the performance evaluation in the next subsections.

	encryption	decryption	addition	left shift and add
GPG	1.7 ms	1.3 ms	N/A	N/A
Paillier	2.5 ms	0.7 ms	7 µs	N/A
XPIR-BV	103 µs	31 µs	3 µs	70 µs

Yao cost	CPU	network transfers
ϕ = integer comparison	71 µs	2501 B
ϕ = argmax	70 µs	3959 B

	map lookup	float addition
NoPriv operations	0.17 µs	0.001 µs

Figure 3.6: Microbenchmarks for operations shared by Pretzel and the baselines (Figure 3.3). Both CPU and network costs are averaged over 1,000 runs; standard deviations (not shown) are within 5% of the averages. OpenPGP encryption and decryption times depend on the length of the email; we use an email size of 75 KB, which is in line with average email size [29]. Similarly, Yao costs for $\phi = \text{argmax}$ depend linearly on the number of input values; we show costs per input value.

3.5.1 Spam filtering

This subsection reports the resource overheads (provider- and client-side CPU time, network transfers, and client-side storage space use) and accuracy of spam filtering in Pretzel.

We set three different values for the number of features in the spam classification model: $N = \{200\text{K}, 1\text{M}, 5\text{M}\}$. These values correspond to the typical number of features in various deployments of Bayesian spam filtering software [9–11]. We also vary the number of features in an email ($L = \{200, 1000, 5000\}$); these values are chosen based on the Ling-spam dataset (average of 377 and a maximum of 3638 features per email) and the Gmail dataset (average of 692 and a maximum of 5215 features per email). The number of categories B is two: spam and non-spam.

Provider-side CPU time. Figure 3.7 shows the per-email CPU time consumed by the provider.

For emails with fewer features ($L = 200$), the CPU time of Pretzel is 2.7× NoPriv's and 0.17× Hybrid's. Pretzel's is more than NoPriv's because in NoPriv the provider does L feature extractions, map lookups, and float additions, which

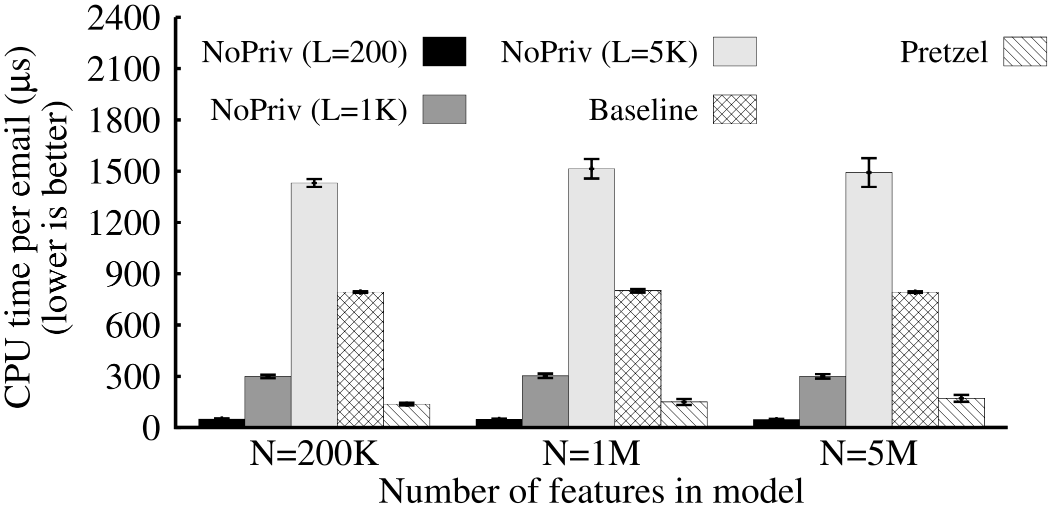

Figure 3.7: Provider-side CPU time per email in microseconds for the spam filtering module while varying the number of features (N) in the spam classification model, and the number of features (L) in an email. CPU time for NoPriv varies only slightly with N (not visible), while (provider-side) CPU times for Hybrid and Pretzel are independent of both L and N (Figure 3.3).

are fast operations (Figure 3.6), whereas in Pretzel, the provider does relatively expensive operations: one additively homomorphic decryption of a XPIR-BV ciphertext plus one comparison inside Yao (Figure 3.3 and §3.3.1). Pretzel's CPU time is lower than Hybrid's because in Pretzel, the provider decrypts a XPIR-BV ciphertext whereas in Hybrid the provider decrypts a Paillier ciphertext (Figure 3.6).

As the number of features in an email increases ($L = \{1000, 5000\}$), provider's CPU time in both Pretzel and Hybrid does not change, as it is independent of L (unlike the client's) while NoPriv's increases since it is linear in L (see Figure 3.3). A particular point of interest is $L = 692$ (the average number of features per email in the Gmail dataset), for which the CPU time of Pretzel is $0.65\times$ NoPriv's (as noted at the beginning of this section, the number is lower than in the status quo in part because Pretzel shifts computational work to the client).

Client-side overheads. Figure 3.8 shows the size of the spam model for the various systems. We notice that the model in Pretzel is approximately $7\times$ smaller than

	Size		
	$N = 200\text{K}$	$N = 1\text{M}$	$N = 5\text{M}$
Non-encrypted	4.3 MB	21.5 MB	107.3 MB
Hybrid	51.6 MB	258.0 MB	1.3 GB
Pretzel-withGLLMPacking	3.1 GB	15.3 GB	76.3 GB
Pretzel	7.4 MB	36.7 MB	183.5 MB

Figure 3.8: Size of encrypted and plaintext spam classification models. N is the number of features in the model. Pretzel-withGLLMPacking is Pretzel, but with the packing in Pretzel replaced with the packing in GLLM (§3.3.2).

the model in Hybrid. This is due to the difference in packing in the two systems: "across rows and columns" (in Pretzel) versus "across columns" (in GLLM [115], implemented in Hybrid (§3.3.2). We also notice that, given the refinement of replacing the cryptosystem (§3.3.1), packing across both rows and columns is essential in Pretzel, to prevent a manifold increase in the model size (the Pretzel-withGLLMPacking row in the figure).

In terms of client-side CPU time, Pretzel takes ≈ 358 ms to process an email with many features ($L = 5000$) against a large model ($N = 5\text{M}$). This time is dominated by the L left shift and add operations in the secure dot product computation (§3.3.2). Our microbenchmarks (Figure 3.6) explain this number: 5000 of the left shift and add operation takes $5000 \times 70\mu\text{s} = 350\text{ms}$. A large L is an unfavorable scenario for Pretzel: client-side processing is proportional to L (Figure 3.3).

Network transfers. Both Pretzel and Hybrid add network overhead relative to NoPriv. It is, respectively, 19.6 KB and 3.6 KB per email (or 26.1% and 4.8% of NoPriv, when considering average email size as reported by [29]). These overheads are due to transfer of a ciphertext and a comparison inside Yao's framework (Figure 3.2). Pretzel's overheads are higher than Hybrid's because the XPIR-BV ciphertext in Pretzel is much larger than the Paillier ciphertext.

Accuracy. Figure 3.9 shows Pretzel's spam classification accuracy for the different classification algorithms it supports. (The figure also shows precision and recall. Higher precision means lower false positives, or non-spam falsely classified as spam; higher recall means lower false negatives, or spam falsely classified as non-spam.)

	Ling-spam			Enron			Gmail		
	Acc.	Prec.	Rec.	Acc.	Prec.	Rec.	Acc.	Prec.	Rec.
GR-NB	99.4	98.1	98.1	98.8	99.2	98.4	98.1	99.7	95.2
LR	99.4	99.4	97.1	98.9	98.4	99.5	98.5	98.9	97.2
SVM	99.4	99.2	97.5	98.7	98.5	99.0	98.5	98.9	97.2
GR	99.3	98.1	97.9	98.8	99.2	98.4	98.1	99.7	95.2

Figure 3.9: Accuracy (Acc.), precision (Prec.), and recall (Rec.) for spam filtering in Pretzel. Sets of columns correspond to the different spam datasets, and the rows correspond to the classification algorithms Pretzel supports: GR-NB, binary LR, and two-class SVM (§3.2.1). Also shown is accuracy for the original Graham-Robinson Naive Bayes algorithm (GR).

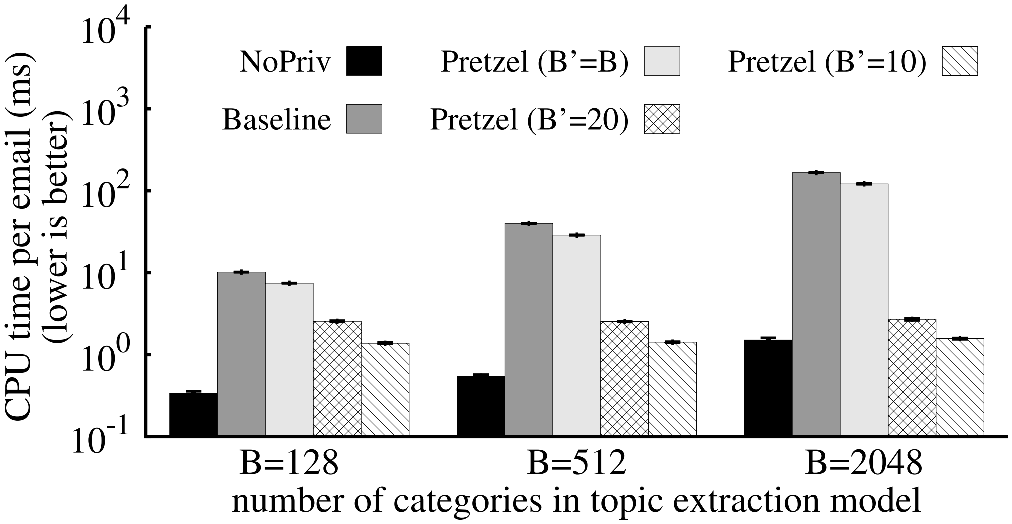

Figure 3.10: Provider-side CPU time per email in milliseconds for topic extraction, varying the number of categories (B) in the model and the number of candidate topics (B'). The case $B = B'$ measures Pretzel without the decomposed classification technique (§3.3.3). The y-axis is log-scaled. N and L are set to 100K and 692 (average number of features per email in the authors' Gmail dataset). The CPU times do not depend on N or L for Pretzel and Hybrid; they increase linearly with L and vary slightly with N for NoPriv.

	network transfers		
	$B = 128$	$B = 512$	$B = 2048$
Hybrid	501.5 KB	2.0 MB	8.0 MB
Pretzel ($B' = B$)	516.6 KB	2.0 MB	8.0 MB
Pretzel ($B' = 20$)	402.0 KB	402.0 KB	401.9 KB
Pretzel ($B' = 10$)	201.0 KB	201.0 KB	201.2 KB

Figure 3.11: Network transfers per email for topic extraction in Pretzel and Hybrid. B' is the number of candidate topics in decomposed classification (§3.3.3). Network transfers are independent of the number of features in the model (N) and email (L) (Figure 3.3).

3.5.2 Topic extraction

This subsection reports the resource overheads (provider- and client-side CPU time, network transfers, and client-side storage space use) and accuracy of topic extraction in Pretzel.

We experiment with $N = \{20\text{K}, 100\text{K}\}$)[7] and $B = \{128, 512, 2048\}$. These parameters are based on the total number of features in the topic extraction datasets we use and Google's public list of topics (2208 topics [123]). For the number of candidate topics for Pretzel (§3.3.3), we experiment with $B' = \{5, 10, 20, 40\}$.

Provider-side CPU time Figure 3.10 shows the per email CPU time consumed by the provider. Without decomposed classification (§3.3.3)—this is the $B' = B$ case in the figure—Pretzel's CPU time is significantly higher than NoPriv's but lower than Hybrid's. Pretzel's time differs from Hybrid's because packed XPIR-BV ciphertexts have lower decryption CPU time per plaintext element than Paillier ciphertexts. With decomposed classification, the number of comparisons inside Yao's framework come down and, as expected, the difference between CPU times in Pretzel and NoPriv drops (§3.3.3). For $B = 2048, B' = 20$, Pretzel's CPU time is $1.78\times$ NoPriv's; for $B = 2048, B' = 10$, it is $1.03\times$ NoPriv's.

Network transfers. Figure 3.11 shows the network transfers per email for Hybrid and Pretzel. As expected, with decomposed classification, Pretzel's network trans-

[7] The number of features in topic extraction models are usually much lower than in spam models because of word variations for spam, for example, FREE and FR33, etc.

	Size	
	$N = 20\text{K}$	$N = 100\text{K}$
Non-encrypted	144.3 MB	769.4 MB
Hybrid	288.4 MB	1.5 GB
Pretzel	720.7 MB	3.8 GB

Figure 3.12: Size of topic extraction models for the various systems. N is the number of features in the model. B is set to 2048.

	Percentage of the total training dataset			
	1%	2%	5%	10%
$B' = 5$	79.6	84.0	90.1	94.0
$B' = 10$	89.6	92.1	95.6	97.7
$B' = 20$	95.9	97.3	98.5	99.3
$B' = 40$	98.7	99.3	99.8	99.9

Figure 3.13: Impact of decomposed classification (§3.3.3) on classification accuracy for the RCV1 dataset with 296 topics. The columns (except the first) correspond to the percentage of the total training dataset used to train the (public) model that extracts candidate topics. The rows correspond to the number of candidate topics (B'). The cells contain the percentage of test documents for which the "true category" (according to a classifier trained on the entire training dataset) is contained in the candidate topics. Higher percentage is better; 100% is ideal.

fers are lower; they are 402 KB per email (or $5.4\times$ the average email size of 75 KB, as reported in [29]) for $B = 2048$, $B' = 20$, and 201 KB per email (or 2.7× the average email size) for $B = 2048$, $B' = 10$.

Client-side overheads. Figure 3.12 shows the model sizes (before feature selection; §3.3.3) for the various systems for different values of N and $B = 2048$. Pretzel's model is bigger than Hybrid's for two reasons. First, its model comprises a public part and an encrypted part that comes from the provider. Second, the ciphertext-to-plaintext size ratio in XPIR-BV is twice that of Paillier.

In terms of client-side CPU time, as in spam filtering, Pretzel (with or without decomposed classification) takes less than half a second to process an email with many features ($L = 5000$).

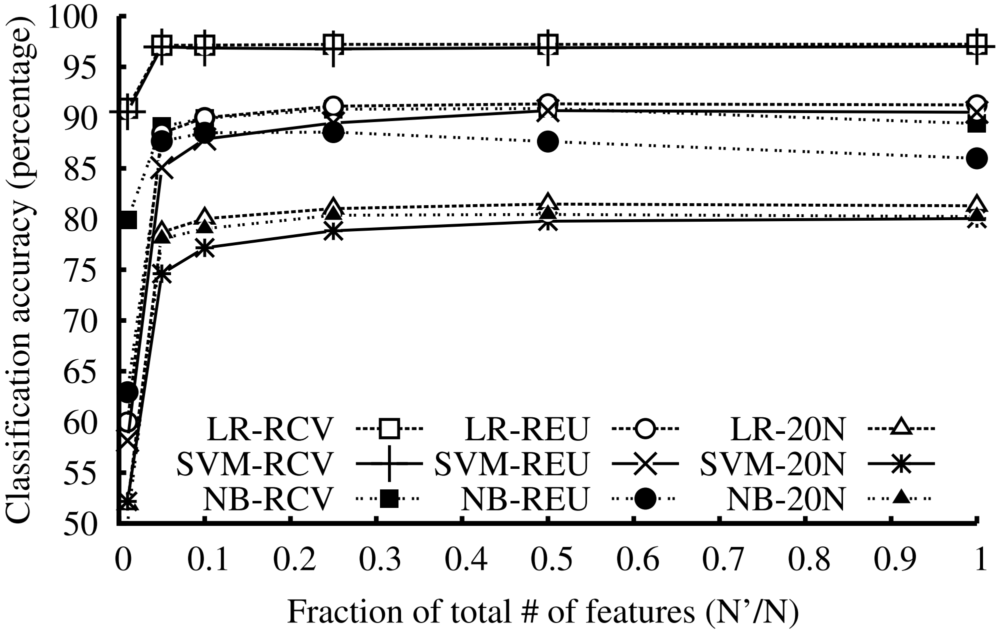

Figure 3.14: Classification accuracy of topic extraction classifiers in Pretzel as a function of N'/N, where N is the total number of features in the training part of the datasets and N' is the number of selected features (§3.3.3). The plotted accuracies are for the 20News (20N), Reuters (REU), and RCV1 (RCV) datasets. 20N and REU come pre-split into training and testing parts: 60%/40% and 75%/25% for the two respectively, whereas we randomly split RCV into 70%/30% training/testing portions. Pretzel can operate at a point where number of features selected N' is roughly 25% of N; this would result in only a marginal drop in accuracy.

Loss of accuracy. Recall that classification accuracy for topic extraction in Pretzel could be affected by decomposed classification and feature selection (§3.3.3). Figure 3.13 shows the variation in classification accuracy due to decomposed classification. (The depicted data are for the RCV1 dataset and NB classifier; the qualitative results are similar for the other datasets and classifiers.) The data suggest that an effective non-proprietary classifier can be trained using a small fraction of training data, for only a small loss in end-to-end accuracy.

Figure 3.14 shows classification accuracy for classifiers trained with and without feature selection, and while varying the degree of feature selection (using the Chi-square selection technique [228]). It appears that even after a high degree of feature selection, accuracy drops only modestly below its peak point. (This would reduce the client-side storage cost presented in Figure 3.12.)

3.5.3 Keyword search and absolute costs

	index size	query time	update time
Ling-spam	5.2 MB	0.32 ms	0.18 ms
Enron	27.2 MB	0.49 ms	0.1 ms
20 Newsgroup	23.9 MB	0.3 ms	0.12 ms
Reuters-21578	6.0 MB	0.28 ms	0.06 ms
Gmail Inbox (40K emails)	50.4 MB	0.13 ms	0.12 ms

Figure 3.15: Client-side search index sizes, CPU times to query a keyword in the indexes (that is, retrieve a list of emails that contain a keyword), and CPU times to index a new email.

Figure 3.15 shows the client-side storage and CPU costs of Pretzel's keyword search module (§3.4).

We now consider whether the preceding costs, in absolute terms, would be acceptable in a deployment. We consider an average user who receives 150 emails daily [239] of average size (75 KB) [29], and owns a mobile device with 32 GB of storage.

To spam filter a long email, the client takes 358 ms, which would be less than a minute daily. As for the encrypted model, one with 5M features occupies 183.5 MB or 0.5% of the device's storage. For network overheads, each email transfers an extra 19.33 KB, which is 2.8 MB daily.

For topic extraction, the client uses less than half a second of CPU per email (or less than 75s daily); a model with 2048 categories (close to Google's) and 20K features occupies 720.7MB or 2.2% of the device's storage (this can be reduced further using feature selection). Also, the client transfers an extra 59 MB (5.4 times the size of the emails) over the network daily, when the number of candidate topics (B') is 20.

Overall, these costs are certainly substantial—and we don't mean to diminish that issue—but we believe that the magnitudes in question are still within tolerance for most users.

3.6 Discussion, limitations, and future work

Pretzel is an improvement over its baseline (§3.2.3) of up to 100×, depending on the resource (§3.5). Its absolute overheads are substantial but, as just discussed (§3.5.3), are within the realm of plausibility.

Pretzel's prototype has several limitations. It handles only the functions we presented (spam filtering, topic extraction, and keyword search) and only using specific algorithms (linear classifiers). Extending Pretzel to include other functions (predictive personal assistance, virus scanning, etc.), other algorithms (neural networks, etc.), or other (potentially cheaper) theoretical machinery [48] is future work. So is adapting Pretzel to hide metadata.

A fundamental limitation of Pretzel is information leakage (§3.1.1). This issue and potential remedies are discussed in Section 3.3.4. To elaborate slightly, providers can protect their models (in the spam function) by periodically revising the model parameters and maintaining different versions for different clients; hiding classifier algorithms, which is another line of future work, would also help [241]. And clients who wish to do so can protect their emails (in topic extraction) by opting out with plausible deniability; also, providers cannot expose all or even a substantial fraction of clients this way, as that would forfeit the original purpose of topic extraction. Nevertheless, defaults being defaults, most clients would probably not opt out, which means that particular clients could indeed be targeted by a sufficiently adversarial provider.

If Pretzel were widely deployed, we would need a way to derive and retrain models. This is a separate problem, with existing research [235, 237, 251, 255–257]; combining Pretzel and this literature is future work.

There are many other obstacles between the status quo and default end-to-end encryption. In general, it's hard to modify a communication medium as entrenched as email [109]. On the other hand, there is reason for hope: TLS between data centers was deployed over just several years [106]. Another obstacle is key management and usability: how do users share keys across devices and find each other's keys? This too is difficult, but there is recent research and commercial attention [8, 55, 180, 244]. Finally, politics: there are entrenched interests who would prefer email not to be encrypted.

Chapter 4

Comparisons and connections

To better understand the design space and rationale for Popcorn and Pretzel, we introduce a framework to situate and compare the various approaches that can provide privacy in online applications such as media consumption and email. Next, we reflect on our choices, and shed light on the fundamental connections between Popcorn and Pretzel.

4.1 Framework

Our framework is depicted in Figure 4.1. It has three axes: trust assumptions, generality, and expense.

4.1.1 Trust assumptions (Axis 1)

Approaches that provide privacy typically make one or more trust assumptions. Examples of such assumptions are *cryptographic hardness*, *anytrust*, and *secure hardware*.

Cryptographic hardness. Cryptographic hardness is the assumption that certain computational problems (factorization of large integers, discrete log, ring-LWE [173], etc.) are unsolvable by an adversary with limited computational capacity. Cryptographic protocols in general assume cryptographic hardness. As an example, Pretzel's 2PC protocol (§3.3) relies on the hardness of the ring-LWE problem.

		generality		
		widely applicable	**a single type of application**	**tailored to one application**
trust assumptions	**crypto. hardness**	FHE [111] (⋆⋆⋆) Yao's 2PC [252] (⋆⋆⋆) GMW 2PC [118] (⋆⋆⋆) OT [60] (⋆⋆⋆)	Yao+GLLM [208] (⋆⋆⋆) CPIR [161] (⋆⋆⋆) GLLM [115] (⋆⋆⋆)	Pretzel's 2PC (⋆)
	anytrust [217]	-	ITPIR [72] (⋆⋆⋆)	-
	anytrust & crypto. hardness	-	-	Popcorn's PIR (⋆⋆)
	secure hardware	Intel SGX [82, 143] (⋆)	hardware-assisted PIR [44, 222] (⋆)	-

Figure 4.1: A framework to situate the various approaches that can hide the content of user requests, especially in online services such as media consumption and email. This framework has three axes. The first axis (left) captures the underlying trust assumptions; the second axis (top) captures generality; and the third axis captures the expense of the approach, denoted by stars within parentheses, where (⋆) denotes low in terms of both consumed resources and dollars, (⋆⋆) denotes high in terms of consumed resources but low in terms of dollars, and (⋆⋆⋆) denotes high in terms of both consumed resources and dollars.

Anytrust. Anytrust refers to the assumption that at least one of the many entities participating in a protocol exactly follows the steps of the protocol; the rest of the entities may deviate arbitrarily from the protocol [217, 248]. Furthermore, one does not need to know which entities correctly follow the protocol. For instance, ITPIR cryptographic protocols (§2.1.4) assume that there are multiple servers that do not all collude with each other.

Cryptographic hardness and anytrust. This assumption combines the two assumptions described above. For example, Popcorn's PIR protocol (§2.3) composes a CPIR (which assumes cryptographic hardness) with an ITPIR protocol (which assumes anytrust).

Secure hardware. Secure hardware refers to the assumption that a hardware component meets a certain set of security-related requirements. For example, Intel's SGX-enabled processors [82, 143] ensure that the memory contents of user-level processes are inaccessible to privileged software like kernel and hypervisor. However, to meet the security-related requirements, Intel SGX additionally depends on authentication services, such as remote attestation [153], run by the hardware manufacturer.

4.1.2 Generality (Axis 2)

The second axis in framework captures how generally applicable an approach is.

Widely applicable. Approaches under this category (second column in Figure 4.1) can be used for a wide variety of applications. One example in this category is Yao's 2PC protocol (§3.2.2), which allows two parties to jointly compute *any* function on their private inputs, without revealing the inputs to each other. This protocol can be used to build private systems for media consumption, email, and other online applications. Another example is fully homomorphic encryption (FHE) [111], which supports *arbitrary* computations in cipherspace.

A single type of application. Unlike widely applicable, approaches in this category deal with only *one type* of application. For example, CPIR, ITPIR, and hardware-assisted PIR [44, 222] protocols apply only to the data retrieval setting. Similarly, Yao+GLLM (§3.2.2) and GLLM [115] protocols apply only to functions that compute dot products.

Tailored. Approaches in this category (rightmost column in Figure 4.1) support typically one application, and have limited scope. For example, Popcorn's PIR protocol applies only to the Netflix-like data retrieval setting where the objects being retrieved are long, similar in size, and consumed progressively. Likewise, Pretzel's 2PC protocol applies only to the linear classification function in which the categories of the classifier are public.

4.1.3 Expense (Axis 3)

This axis captures two different but related costs: resource consumption (CPU, network, and disk I/O) of an approach, and the resource consumption converted into a dollar amount.

Low in terms of both consumed resources and dollars. Approaches that fall in this category have a low resource overhead, which translates to a small dollar cost. Pretzel's 2PC protocol falls in this category. Approaches that use Intel SGX also have low resource overheads because they run unmodified (or slightly modified) application code.

High in terms of consumed resources but low in terms of dollars. Approaches that heavily consume only the cheap resources fall in this category. For instance, Popcorn's PIR protocol consumes CPU (a cheap resource) extensively but is engineered to conserve the more expensive resources (disk I/O and network).

High in terms of both consumed resources and dollars. Many approaches consume all (or at least the expensive) resources heavily. For example, Yao's 2PC protocol performs cryptographic operations for individual gates in a large Boolean circuit, resulting in a high CPU and network cost (§3.2.2). Similarly, CPIR and ITPIR protocols require operating on the entire object library (movie library, etc.), resulting in a high CPU and disk I/O overhead.

4.2 Our choices

In our framework (§4.1), Popcorn's PIR protocol is represented by (cryptographic hardness and anytrust, tailored, high in terms of consumed resources but low in terms of dollars), and Pretzel's 2PC protocol is represented by (cryptographic hardness, tailored, low in terms of both consumed resources and dollars). We now explain why these choices are appropriate in our context (§1.1).

For the first axis, we prefer cryptographic hardness and anytrust based assumptions over secure hardware, for two reasons.

First, we did not want to put trust into a single entity whereas the secure hardware assumption demands that the entity designing or manufacturing the hardware (for example, Intel) is trusted. In contrast, anytrust demands trusting one of the participating entities; not a particular one. Further, cryptographic hardness requires trusting hardness of one or more well-studied computational problems. Second, approaches under the cryptographic hardness and anytrust assumptions are based on solid theoretical foundations and make reasoning about privacy properties easier. In contrast, systems built on trusted hardware can have several side-channels (cache access patterns, memory access patterns, power draw, etc.) through which private information can leak.

For the second axis, we could choose any of the three categories. Approaches under “widely applicable” and “a single type of application” categories can support sophisticated functions besides basic functions. However, we choose the “tailored” category for both Popcorn and Pretzel to fulfill our requirement of supporting only basic functions (§1.1).

For the third axis, the category “low in terms of both consumed resources and dollars” is a natural choice. We choose this category for Pretzel. For Popcorn, we cannot choose the same category. There are three approaches—Pretzel’s 2PC, Intel SGX, and hardware-assisted PIR—under this category. Pretzel’s 2PC is not applicable to Netflix-like media delivery since it is tailored to the email setting. The other two approaches are in conflict with our requirement of providing strong privacy guarantees (§1.1). Therefore, for Popcorn we choose the next best category of “high in terms of consumed resources but low in terms of dollars”.

4.3 Connections between Popcorn and Pretzel

Popcorn and Pretzel, although built for different applications, have significant connections. In this section, we elaborate on the connections, similarities, and differences between these systems.

Popcorn and Pretzel exhibit a connection by sharing the same design ethos of tailoring cryptographic protocols to the application. Popcorn takes PIR cryptographic protocols and tailors them for the Netflix-like media delivery setting (§2.3). Likewise, Pretzel starts from a 2PC protocol and tailors it to the email setting (§3.3).

While designing Popcorn and Pretzel, we uncovered another connection as

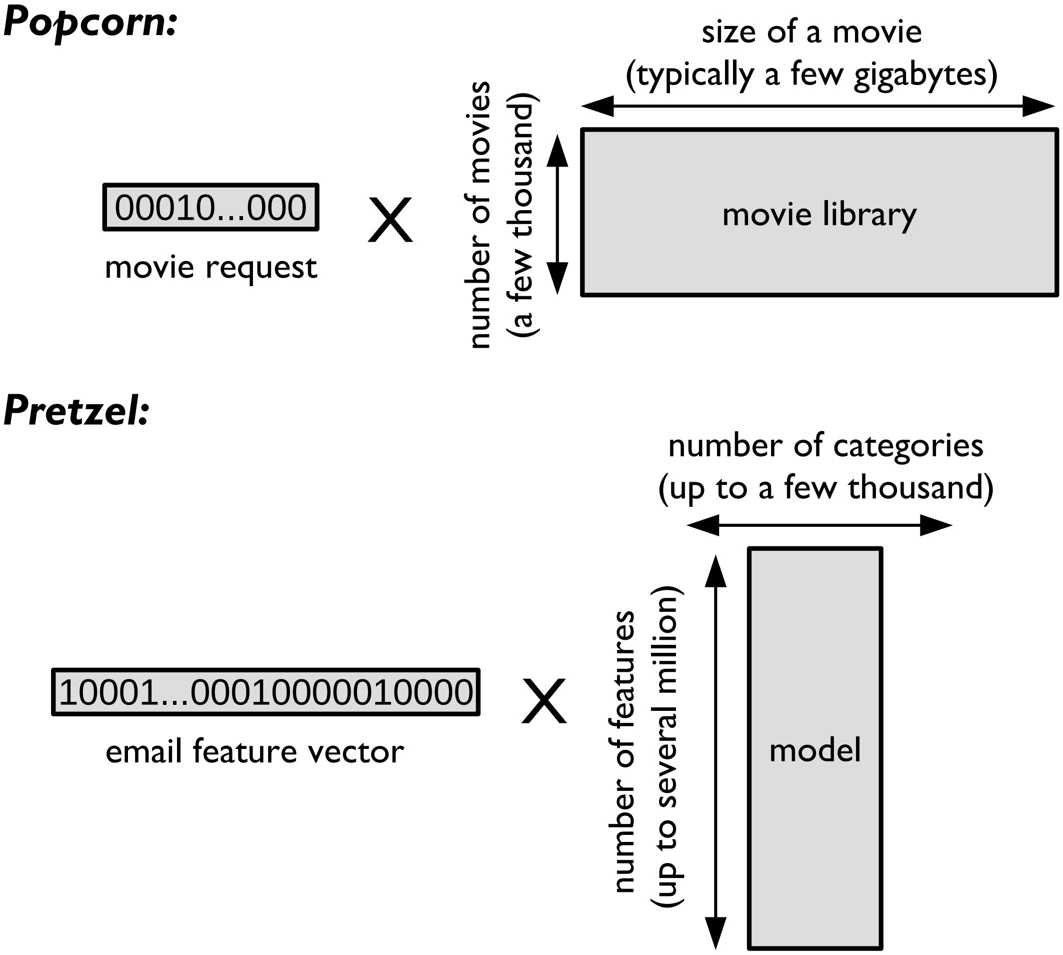

Figure 4.2: Vector-matrix product in Popcorn (top) and Pretzel (bottom). Popcorn's matrix is several tens of terabytes in size, while Pretzel's matrix is only a few hundred megabytes. The vector in Popcorn is much shorter than in Pretzel.

both these systems compute a vector-matrix product. The vector encodes a movie or an email, and the matrix encodes a movie library or a classification model. To compute the vector-matrix product privately, both systems build on additively homomorphic encryption.

We also observed differences in the vector and matrix dimensions while designing these systems (Figure 4.2). In Popcorn, the vector is short (a few thousand elements) and the matrix is large (a few tens of terabytes in size). In contrast, the vector is long (a few million elements) and the matrix is small (a few hundred megabytes in size) in Pretzel. These differences in vector and matrix dimensions influence the designs of the two systems.

- In Popcorn, the server stores the matrix as clients don't have terabytes of storage, and a client encrypts and sends a short vector to the server. In contrast, in Pretzel, a client keeps the long vector, and the server encrypts and sends the matrix to the client. Thus, in Popcorn, the bulk of the computation is performed at the server, while in Pretzel it is performed at the client.

- Popcorn demands a solution for high disk I/O costs, for two reasons. First, its matrix is large in size. Second, an encrypted vector is dense, and vector density is proportional to the fraction of the matrix that has to be touched to compute the vector-matrix product (§2.3.2). In Pretzel, high disk I/O is not an issue, as its vector is sparse (§3.5).

Besides the connections described above, there is an implicit connection between Popcorn and Pretzel worth mentioning. While designing Popcorn, we came across XPIR [35], a prior work on private Netflix-like media delivery, and noticed the low decryption cost of its underlying additively homomorphic encryption scheme (XPIR-BV). While building Pretzel, when we were looking for a cryptosystem that could meet the two requirements of low decryption cost, and additively homomorphic properties, XPIR-BV was an obvious choice (§3.3.1). Popcorn has facilitated Pretzel.

Chapter 5

Summary, lessons learned, discussion

In this dissertation, we studied how one could build a Netflix-like media delivery and an email service such that they (a) provide strong privacy guarantees to the users, (b) support the provider's basic functions, and (c) have affordable resource costs. To satisfy these requirements, our first step was to turn to cryptographic protocols (private information retrieval and secure two-party computation). Although these protocols met the first two requirements, there was a key challenge: huge resource costs.

We addressed this challenge by following the design ethos of tailoring cryptographic protocols using application-specific properties. Based on this design ethos, we built two systems—Popcorn and Pretzel. Popcorn is a Netflix-like media delivery system that provably hides, from both the service provider and a network eavesdropper, which movie a user is watching. It relies on private information retrieval (PIR) cryptographic protocols (§2.1.2). It composes two different types of PIR to ease a tension between overheads and content protection (§2.3.1), batches requests from a large number of users to amortize the overheads of PIR (§2.3.2), forms large batches while keeping a small playback delay by leveraging streaming (§2.3.3), and uses a combination of padding and bitrate reduction to handle variations in media object sizes (§2.3.4). The cost of serving an object in Popcorn, in terms of dollars spent, is 3.87× that of a non-private system (§2.5). In contrast, the factor is 265 if PIR is applied as it is described in the literature.

The other system described in this dissertation, Pretzel, is an email system in which email contents are encrypted end-to-end between senders and intended recipients, and the email service provider performs spam filtering and topic extraction over the email contents, using linear classifiers from machine learning (Naive Bayes, Logistic regression, and SVMs; §3.2.1). Pretzel relies on a secure two-party computation cryptographic protocol (§3.2.3). It reduces the resource consumption of this protocol by replacing the underlying cryptosystem (§3.3.1), conserving calls into the cryptosystem (§3.3.2), and decomposing classification into a private and non-private step (§3.3.3). Pretzel's overheads, relative to a non-private implementation, are up to $5.4\times$ for the provider, with additional but tolerable client-side requirements (§3.5). These costs represent reductions versus the starting 2PC protocol of between $1.8\times$ and $100\times$ (§3.5).

Popcorn and Pretzel are fundamentally connected (§4.3). They both compute a vector-matrix product privately, however, the dimensions of the vectors and matrices in the two are different. A consequence is that the designs of the two systems have to address different challenges: high disk I/O in Popcorn versus high network costs in Pretzel.

Both Popcorn and Pretzel have several limitations; some of these limitations are fundamental, while others are either tied to the design or the prototype (§2.7, §3.6). Nevertheless, they demonstrate that it is possible to build in strong privacy properties into commonly used online services.

Lessons learned. We learned several lessons while building Popcorn and Pretzel.

- *A few compromises on design requirements can enable progress.* To build Popcorn we decided to avoid ITPIR (§2.1.4) at first because ITPIR assumes non-colluding servers; instead, we attempted to rely only on CPIR (§2.1.3). Although CPIR makes weaker assumptions than ITPIR, it is much more expensive; as a result, we were not able to build a practical system for many months. However, after compromising on the set of assumptions, and composing CPIR with ITPIR, we were able to take significant strides in reducing costs.

  Similarly, our first attempt to build Pretzel used sophisticated non-linear classifiers. Again, we were not able to make progress until we switched to linear classifiers with a simpler structure. In hindsight, the switch appears to be a good choice because not only did we manage to leverage the simple structure to move swiftly, but also the sacrifice in accuracy, due to the reduction in sophistication, ended up being tolerable (§3.5).

- *Cryptographic protocols can be your best friends.* On the surface, cryptographic protocols like PIR and 2PC appear scary (at least to systems researchers); they have a steep learning curve and they are, if directly applied, impractical, due to their huge resource costs. For example, PIR protocols entail a server-side overhead that is linear in the size of the server's library. Nevertheless, these protocols have certain advantages. First, they are based only on cryptographic hardness assumptions rather than on assumptions that require trusting a single entity. Second, they are accompanied by rigorous proofs and make reasoning about the end-to-end guarantees of a system easier. For example, in Pretzel, using the guarantees of 2PC protocols, we were able to at least say that the end-to-end leakage is bounded in the number of bits.
- *Knowledge gained while building one system can facilitate the design of another system.* While working on Popcorn, we learned about additively homomorphic encryption (AHE) schemes proposed for a private Netflix-like media delivery application. Further, we learned about protocols that can securely compute vector-matrix products. This knowledge proved to be very useful when we were building Pretzel. First, when we learned that Pretzel needed to compute a vector-matrix product we were able to turn to the protocols that we had studied earlier. Second, Pretzel needed an AHE scheme that had a low decryption CPU time. Here, we were able to use the exact scheme that was proposed for the private Netflix-like application.
- *Explore different designs to restrict unavoidable leakage.* Sometimes it is not possible to compute over user private data without avoiding leakage. For instance, performing topic extraction over a private email leads to leakage of one or more topics. In such scenarios, leakage of topics can be restricted, by exploring different designs. Our first attempt to build Pretzel[1] leaked many topics per email. However, using an alternate design (§3.3.3), we were able to restrict the leakage to a single topic.

Discussion. We demonstrated how to build private and plausibly deployable systems for on-demand media delivery (Popcorn) and email (Pretzel). To move further toward a real-world deployment of these systems, future research focus should be (a) to expand on the functions they support, and (b) to improve the privacy properties they provide.

[1] https://arxiv.org/pdf/1612.04265v2.pdf

Popcorn will gain from support for movie recommendations. One could start this pursuit by trying to compose relevant prior work on privacy-preserving recommendations [66, 150, 157] with Popcorn.

For its part, Pretzel will benefit from two additions to improve its privacy properties. First is support for proprietary algorithms; in principle, one can use 2PC to hide the service provider's proprietary algorithms, but the challenge will be to tame the high resource costs. Second is support for hiding email metadata; there are prior works [41, 80, 162, 238] that can provably hide the metadata of messaging applications but, again, the resource costs of these works are too high for the email setting. Adapting techniques from these works into a provably-secure low-cost system for hiding email metadata could be a promising approach.

Looking forward, Popcorn and Pretzel may not completely replace the status quo, even after extensions, for different reasons. First, decision makers may prefer surveillance and mass data collection (§3.6). Second, some users may not want to trade-off functionality for strong privacy properties. However, we believe that Popcorn and Pretzel can at least be an alternative to the status quo, especially for the people who do not want to compromise on their privacy. These systems can also motivate future research on building private and practical systems for other online applications like web search. Our hope is that the systems we built, and the ones they will shape, will impact discourse about the viability of a world where people's right to privacy on the web is not left solely in the hands of laws, policies, trust, and hope, but vigorously defended through technical solutions.

Appendix A

Derivation of segment sizes in Popcorn

Recall the inequalities defined in Section 2.3.3:

$$t_i \leq T_i \cdot \alpha, \quad \text{where } \alpha = \frac{\min\{R_i, P_i/b_i\}}{\mu \cdot n}$$

$$T_i \leq d' + \sum_{j=1}^{i-1} t_j, \quad \text{where } d' = d - \epsilon$$

We consider the special case where both sides of the inequalities are equal. Combining both statements:

$$t_i = \left(d' + \sum_{j=1}^{i-1} t_j \right) \cdot \alpha$$

We show that $t_i = (d' \cdot \alpha \cdot (1+\alpha)^{i-1})$ is a solution to the above equation. Substituting on both sides:

$$d' \cdot \alpha \cdot (1+\alpha)^{i-1} = \left(d' + \sum_{j=1}^{i-1} d' \cdot \alpha \cdot (1+\alpha)^{j-1} \right) \cdot \alpha.$$

Summing the finite geometric series, and rearranging:

$$= \left(d' + d' \cdot \alpha \cdot \left(\frac{(1+\alpha)^{i-1} - 1}{\alpha} \right) \right) \cdot \alpha$$

$$= d' \cdot \alpha \cdot (1+\alpha)^{i-1}.$$

Setting $\alpha = 1$, we get $t_i = 2^{i-1} \cdot (d - \epsilon)$, as desired.

Appendix B

Details on Popcorn's pricing model

Our high-level goal is to estimate the hourly cost of renting three resources on Amazon EC2: a vCPU, 1 GB of memory, and 1 Gbps of sequential read I/O bandwidth. To get the estimates, we make the simplifying assumption that the price of an EC2 machine depends only on these three resources. Of course, in practice, pricing machines is a complex process that depends on many factors (I/O performance for non-sequential workloads, cost of the networking infrastructure, prices set by competitors, etc.); the values derived here should be treated as only estimates.

At a high level, our method is to use the specification of machines on Amazon EC2 and their corresponding prices to derive a system of linear equations; in these equations variables represent the unit cost of the resources mentioned above, coefficients represent the "quantity" of those resources in an Amazon EC2 machine, and the RHS will be the price of renting that machine.

We consider the machines in Figure 2.8 and an additional machine. We need this additional machine as the equation for i2.4xl is not linearly independent from that of i2.8xl, which leaves us with two equations to solve for three variables. To write the third equation, we pick a memory optimized machine that has 32 vCPUs, 244 GB of memory capacity, 2 SSDs with 320 GB capacity each (6.4 Gbps sequential read I/O bandwidth), and is rented out for $0.9822 per hour. Using these, we get the following equations:

$$\begin{aligned} 32C + 60M + 6.4I &= 0.6281 \\ 32C + 244M + 6.4I &= 0.9822 \\ 32C + 244M + 23.3I &= 1.6902, \end{aligned}$$

where C is the hourly cost of renting a vCPU, M is the cost of renting 1 GB of memory for an hour, and I is the hourly cost for 1 Gbps of sequential read I/O bandwidth.

Solving for the unknowns in the equations, we get I=0.042, M=0.0019, and C=0.0076.

Appendix C

Details of Naive Bayes

C.1 Spam classification

Section 3.2.1 stated that the GR-NB algorithm labels a document (email) as spam if $\alpha = 1/p(\text{spam} \mid \vec{x}) - 1$ is greater than some fixed threshold, and that $\log \alpha$ can be expressed as

$$\left(\sum_{i=1}^{i=N} x_i \cdot \log p(t_i \mid C_2)\right) + 1 \cdot \log p(C_2)$$
$$- \left(\sum_{i=1}^{i=N} x_i \cdot \log p(t_i \mid C_1)\right) + 1 \cdot \log p(C_1), \tag{C.1}$$

where the spam category is represented by C_1 and non-spam is C_2. Here we show this derivation.

From Bayes Rule:

$$p(C_1 \mid \vec{x}) = \frac{p(\vec{x} \mid C_1) \cdot p(C_1)}{p(\vec{x})}.$$

Rewriting:

$$\frac{1}{p(C_1 \mid \vec{x})} - 1 = \frac{p(\vec{x})}{p(\vec{x} \mid C_1) \cdot p(C_1)} - 1.$$

Expanding $p(\vec{x})$ to $p(\vec{x} \mid C_1) \cdot p(C_1) + p(\vec{x} \mid C_2) \cdot p(C_2)$:

$$= \frac{p(\vec{x} \mid C_2) \cdot p(C_2)}{p(\vec{x} \mid C_1) \cdot p(C_1)}. \tag{C.2}$$

The Naive Bayes assumption says that the presence (or absence) of a feature is independent of the other features, which means

$$p(\vec{x} \mid C_j) = \prod_{i=1}^{i=N} p(t_i \mid C_j)^{x_i}.$$

Plugging the above equation into equation (C.2):

$$\alpha = \frac{\prod_{i=1}^{i=N} p(t_i \mid C_2)^{x_i} \cdot p(C_2)}{\prod_{i=1}^{i=N} p(t_i \mid C_1)^{x_i} \cdot p(C_1)}.$$

In log space, the expression above is equation (C.1).

C.2 Multinomial text classification

Section 3.2.1 stated that the multinomial NB algorithm identifies the category C_{j^*} that maximizes $j^* = \text{argmax}_j \, p(C_j \mid \vec{x})$. The section stated that the maximal $p(C_j \mid \vec{x})$ can be determined by examining

$$\left(\sum_{i=1}^{i=N} x_i \cdot \log p(t_i \mid C_j) \right) + 1 \cdot \log p(C_j),$$

for each j. Here we give a derivation of this fact.

From Bayes Rule:

$$p(C_j \mid \vec{x}) = \frac{p(\vec{x} \mid C_j) \cdot p(C_j)}{p(\vec{x})}. \tag{C.3}$$

The multinomial Naive Bayes algorithm assumes that a document, email, or feature vector $\vec{x}$ is formed by picking, with replacement, $L = \sum_{i=1}^{i=N} x_i$ features from the set of N features in the model. Then, $p(\vec{x} \mid C_j)$ is modeled as a multinomial distribution:

$$p(\vec{x} \mid C_j) = \frac{p(L) \cdot L!}{\prod_{i=1}^{i=N} x_i!} \cdot \prod_{i=1}^{i=N} p(t_i \mid C_j)^{x_i}.$$

Plugging the above equation into Equation (C.3):

$$= \frac{1}{p(\vec{x})} \cdot \frac{p(L) \cdot L!}{\prod_{i=1}^{i=N} x_i!} \cdot \prod_{i=1}^{i=N} p(t_i \mid C_j)^{x_i} \cdot p(C_j)$$

Discounting the terms that are independent of j (which we are free to do because the maximum is unaffected by such terms), the expression above becomes:

$$\prod_{i=1}^{i=N} p(t_i \mid C_j)^{x_i} \cdot p(C_j),$$

which, when converted to log space, is the claimed expression.

Bibliography

[1] Alphabetical List - Fri, Apr 3, 2015. http://usa.netflixable.com/2015/04/alphabetical-list-fri-apr-3-2015.html.

[2] Apache SpamAssassin. http://spamassassin.apache.org/.

[3] Digital Rights Management. http://msdn.microsoft.com/en-us/library/cc838192%28VS.95%29.aspx.

[4] Enron email dataset. https://www.cs.cmu.edu/~./enron/.

[5] Free haven's selected papers in anonymity. http://freehaven.net/anonbib/.

[6] GPGME. https://www.gnupg.org/software/gpgme/index.html.

[7] Home page for 20 Newsgroups data set. http://qwone.com/~jason/20Newsgroups/.

[8] Keybase. https://keybase.io.

[9] Large-scale global Bayes tuning? http://users.spamassassin.apache.narkive.com/d6ppUDfw/large-scale-global-bayes-tuning.

[10] Mailing List Archive: Bayes db and token expiry questions. http://www.gossamer-threads.com/lists/spamassassin/users/151578.

[11] Mail::SpamAssassin::Conf - SpamAssassin configuration file. http://spamassassin.apache.org/full/3.4.x/doc/Mail_SpamAssassin_Conf.html.

[12] Microsoft PlayReady. http://www.microsoft.com/playready/.

[13] Netflix 2015 Q1 Earnings Letter. http://files.shareholder.com/downloads/NFLX/47469957x0x821407/DB785B50-90FE-44DA-9F5B-37DBF0DCD0E1/Q1_15_Earnings_Letter_final_tables.pdf.

[14] Netflix Soars On Subscriber Growth. http://www.forbes.com/sites/laurengensler/2015/01/20/netflix-soars-on-subscriber-growth/.

[15] New Movie Arrivals - Fri, Apr 3, 2015. http://usa.netflixable.com/2015/04/new-movie-arrivals-fri-apr-3-2015.html.

[16] OpenPGP. http://openpgp.org/.

[17] Protecting Consumer Privacy in an Era of Rapid Change: Recommendations for Business and Policy-makers. http://www.ftc.gov/sites/default/files/documents/reports/federal-trade-commission-report-protecting-consumer-privacy-era-rapid-change-recommendations/120326privacyreport.pdf.

[18] Pyramid broadcast technique for video on demand. Lecture notes, http://www.mathcs.emory.edu/~cheung/Courses/558-old/Syllabus/5-VoD/pyramid.html.

[19] REGULATION OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL on the protection of individuals with regard to the processing of personal data and on the free movement of such data (general data protection regulation). http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=COM:2012:0011:FIN:EN:PDf.

[20] Reuters-21578 text categorization test collection. http://www.daviddlewis.com/resources/testcollections/reuters21578/.

[21] scikit-learn: Machine learning in Python. http://scikit-learn.org/stable/.

[22] SpamBayes: Bayesian anti-spam classifier written in Python. http://spambayes.sourceforge.net/.

[23] SQLite FTS3 and FTS4 extensions. https://www.sqlite.org/fts3.html.

[24] The 2014 Pulitzer Prize Winners, Public Service: The Guardian US and The Washington Post. http://www.pulitzer.org/works/2014-Public-Service.

[25] The WebM Project. http://www.webmproject.org/about/faq/.

[26] WebM Encryption. http://www.webmproject.org/docs/webm-encryption/.

[27] Weka 3: Data mining software in Java. http://www.cs.waikato.ac.nz/ml/weka/.

[28] Welcome to SpamProbe - A fast Bayesian spam filter. http://spamprobe.sourceforge.net/.

[29] What is the average size of an email message? `http://email.about.com/od/emailstatistics/f/What_is_the_Average_Size_of_an_Email_Message.htm`.

[30] You are watching more web video ads than ever. `http://allthingsd.com/20130215/you-are-watching-more-web-video-ads-than-ever/`.

[31] A survey on ring-LWE cryptography, Feb. 2016. `https://www.microsoft.com/en-us/research/video/a-survey-on-ring-lwe-cryptography/`.

[32] V. K. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z.-L. Zhang. Unreeling Netflix: Understanding and improving multi-CDN movie delivery. In *IEEE International Conference on Computer Communications (INFOCOM)*, 2012.

[33] P. Aditya, R. Sen, P. Druschel, S. J. Oh, R. Benenson, M. Fritz, B. Schiele, B. Bhattacharjee, and T. T. Wu. I-Pic: A platform for privacy-compliant image capture. In *International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2016.

[34] C. Aguilar-Melchor, J. Barrier, L. Fousse, and M.-O. Killijian. XPIR: Private Information Retrieval for Everyone. Cryptology ePrint Archive, Report 2014/1025, 2014.

[35] C. Aguilar-Melchor, J. Barrier, L. Fousse, and M.-O. Killijian. XPIR: Private Information Retrieval for Everyone. In *Privacy Enhancing Technologies Symposium (PETS)*, 2016.

[36] C. Aguilar-Melchor and P. Gaborit. A lattice-based computationally-efficient private information retrieval protocol. In *Western European Workshop on Research in Cryptology (WEWoRC)*, 2007.

[37] W. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2001.

[38] O. M. Alliance. DRM Architecture. `http://technical.openmobilealliance.org/Technical/release_program/docs/DRM/V2_1-20081106-A/OMA-AD-DRM-V2_1-20081014-A.pdf`, Mar. 2004.

[39] A. Amirbekyan and V. Estivill-Castro. A new efficient privacy-preserving scalar product protocol. In *Australasian conference on Data mining and analytics (AusDM)*, 2007.

[40] I. Androutsopoulos, J. Koutsias, K. Chandrinos, G. Paliouras, and C. Spyropoulos. An evaluation of Naive Bayesian anti-spam filtering. In *Workshop on Machine Learning in the New Information Age*, 2000.

[41] S. Angel and S. T. Setty. Unobservable communication over fully untrusted infrastructure. In *OSDI*, 2016.

[42] Ann Cavoukian. Privacy by Design. The 7 Foundational Principles. `http://www.privacybydesign.ca/content/uploads/2009/08/7foundationalprinciples.pdf`.

[43] Ann Cavoukian. Privacy by Design in Law, Policy and Practice. a White Paperfor Regulators, Decision-makers and Policy-makers, August 2011. `https://www.ipc.on.ca/images/Resources/pbd-law-policy.pdf`.

[44] D. Asonov and J.-C. Freytag. Almost optimal private information retrieval. In *Workshop on Privacy Enhancing Technologies (PET)*, 2003.

[45] M. L. G. at National Taiwan University. LIBLINEAR–A library for large linear classification. `https://www.csie.ntu.edu.tw/~cjlin/liblinear/`.

[46] M. J. Atallah and W. Du. Secure multi-party computational geometry. In *Workshop on Algorithms and Data Structures (WADS)*, 2001.

[47] M. Backes, A. Kate, M. Maffei, and K. Pecina. ObliviAd: Provably secure and practical online behavioral advertising. In *IEEE Symposium on Security and Privacy (S&P)*, 2012.

[48] M. Ball, T. Malkin, and M. Rosulek. Garbling gadgets for boolean and arithmetic circuits. In *ACM Conference on Computer and Communications Security (CCS)*, 2016.

[49] E. Balsa, C. Troncoso, and C. Diaz. OB-PWS: Obfuscation-based private web search. In *IEEE Symposium on Security and Privacy (S&P)*, 2012.

[50] D. Beeby. Rogue tax workers snooped on ex-spouses, family members. *Toronto Star*, June 2010. `https://www.thestar.com/news/canada/2010/06/20/rogue_tax_workers_snooped_on_exspouses_family_members.html`.

[51] A. Beimel, Y. Ishai, and T. Malkin. Reducing the servers' computation in private information retrieval: PIR with preprocessing. *Journal of Cryptology*, 17(2):125–151, 2004.

[52] E. Betters. What is Google Assistant, how does it work, and when can you use it?, Sept. 2016. `http://www.pocket-lint.com/news/137722-what-is-google-assistant-how-does-it-work-and-when-can-you-use-it`.

[53] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*, 2013.

[54] M. Blanton and P. Gasti. Secure and efficient protocols for iris and fingerprint identification. In *European Symposium on Research in Computer Security (ESORICS)*, 2011.

[55] J. Bonneau. EthIKS: Using Ethereum to audit a CONIKS key transparency log. In *Financial Cryptography and Data Security Conference (FC)*, 2016.

[56] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Workshop on Computational Learning Theory (COLT)*, 1992.

[57] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. In *Network and Distributed System Security Symposium (NDSS)*, 2014.

[58] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):13, 2014.

[59] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *Advances in Cryptology—CRYPTO*, 2011.

[60] G. Brassard, C. Crepeau, and J.-M. Robert. All-or-nothing disclosure of secrets. In *Advances in Cryptology—CRYPTO*, 1987.

[61] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees*. CRC press, 1984.

[62] J. Bringer, O. El Omri, C. Morel, and H. Chabanne. Boosting GSHADE capabilities: New applications and security in malicious setting. In *Symposium on Access Control Models and Technologies (SACMAT)*, 2016.

[63] M. Burkhart and X. A. Dimitropoulos. Fast privacy-preserving top-k queries using secret sharing. In *International Conference on Computer Communication Networks (ICCCN)*, 2010.

[64] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. Openpgp message format. RFC 4880, Internet Engineering Task Force (IETF), 2007.

[65] J. Camenisch, M. Dubovitskaya, and G. Neven. Unlinkable priced oblivious transfer with rechargeable wallets. In *International Conference on Financial Cryptography and Data Security (FC)*, 2010.

[66] J. Canny. Collaborative filtering with privacy. In *IEEE Symposium on Security and Privacy (S&P)*, 2002.

[67] J. Cappos. Avoiding theoretical optimality to efficiently and privately retrieve security updates. In *International Conference on Financial Cryptography and Data Security (FC)*, 2013.

[68] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *Network and Distributed System Security Symposium (NDSS)*, 2014.

[69] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology—CRYPTO*, 2013.

[70] Y.-T. Chiang, D.-W. Wang, C.-J. Liau, and T.-s. Hsu. Secrecy of two-party secure computation. In *Working Conference on Data and Applications Security (DBSec)*. 2005.

[71] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Symposium on Foundations of Computer Science (FOCS)*, 1995.

[72] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.

[73] P. Ciano. How to use Google Now, Mar. 2014. `https://paulciano.org/2014/03/getting-google-now/`.

[74] M. Cohen. Web storage overview. `https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/`.

[75] K. Conger. Google engineer says he'll push for default end-to-end encryption in Allo, May 2016. `https://techcrunch.com/2016/05/19/google-engineer-says-hell-push-for-default-end-to-end-encryption-in-allo/`.

[76] K. Conger. Google's Allo won't include end-to-end encryption by default, May 2016. `https://techcrunch.com/2016/05/18/googles-allo-wont-include-end-to-end-encryption-by-default/`.

[77] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of symbolic computation*, 9(3):251–280, 1990.

[78] J. Corbet. The grumpy editor's guide to bayesian spam filters, 2006. `https://lwn.net/Articles/172491/`.

[79] G. V. Cormack. TREC 2007 spam track overview. In *Text REtrieval Conference (TREC)*, 2007.

[80] H. Corrigan-Gibbs, D. Boneh, and D. Mazières. Riposte: An anonymous messaging system handling millions of users. In *IEEE Symposium on Security and Privacy (S&P)*, 2015.

[81] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[82] V. Costan and S. Devadas. Intel SGX explained. Cryptology ePrint Archive, Report 2016/086, 2016.

[83] W. Dai, Y. Doröz, and B. Sunar. Accelerating SWHE based PIRs using GPUs. Cryptology ePrint Archive, Report 2015/462, 2015.

[84] A. Davis, J. Parikh, and W. E. Weihl. Edgecomputing: Extending enterprise applications to the edge of the internet. In *International World Wide Web conference on Alternate track papers & posters (WWW Alt.)*, 2004.

[85] R. De Clercq, S. S. Roy, F. Vercauteren, and I. Verbauwhede. Efficient software implementation of ring-LWE encryption. In *Design, Automation & Test in Europe (DATE)*, 2015.

[86] D. Demmler, A. Herzberg, and T. Schneider. RAID-PIR: Practical multi-server PIR. In *Cloud computing security workshop (CCSW)*, 2014.

[87] Y. G. Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–458, 1994.

[88] C. Devet. Evaluating private information retrieval on the cloud. Technical Report 5, University of Waterloo, 2013.

[89] C. Devet and I. Goldberg. The best of both worlds: Combining information-theoretic and computational PIR for communication efficiency. In *Privacy Enhancing Technologies Symposium (PETS)*, 2014.

[90] C. Devet, I. Goldberg, and N. Heninger. Optimally robust private information retrieval. In *USENIX Security Symposium (SEC)*, 2012.

[91] G. Di Crescenzo, T. Malkin, and R. Ostrovsky. Single database private information retrieval implies oblivious transfer. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2000.

[92] T. Dierks and E. Rescorla. The transport layer security (TLS) protocol version 1.2. RFC 5246, Network Working Group, Aug. 2008.

[93] W. Diffie and M. Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[94] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium (SEC)*, 2004.

[95] Discretix Technologies. Secure implementations of content protection DRM schemes on consumer electronic devices. `http://www.discretix.com/wp-content/uploads/2013/02/secure_implementation_of_content_protection_schemes_on_consumer_electronic_devices.pdf`, Feb. 2013.

[96] J. Dizon. Gmail can now automatically put flight, hotel, ticket, or restaurant info on Google calendar, Aug. 2015. `http://www.techtimes.com/articles/79380/20150826/gmail-can-now-automatically-put-flight-hotel-ticket-or-restaurant-info-on-google-calendar.htm`.

[97] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the impact of video quality on user engagement. *ACM SIGCOMM Computer Communication Review*, 41(4):362–373, 2011.

[98] Y. Dodis, R. Gennaro, J. Håstad, H. Krawczyk, and T. Rabin. Randomness extraction and key derivation using the CBC, cascade and HMAC modes. In *Advances in Cryptology—CRYPTO*, 2004.

[99] J. Domingo-Ferrer, A. Solanas, and J. Castellà-Roca. h(k)-private information retrieval from privacy-uncooperative queryable databases. *Online Information Review*, 33(4):720–744, 2009.

[100] C. Dong and L. Chen. A fast secure dot product protocol with application to privacy preserving association rule mining. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2014.

[101] C. Dong and L. Chen. A fast single server private information retrieval protocol with low communication cost. In *European Symposium on Research in Computer Security (ESORICS)*, 2014.

[102] W. Du and M. J. Atallah. Protocols for secure remote database access with approximate matching. In *E-Commerce Security and Privacy*, 2001.

[103] W. Du and Z. Zhan. Building decision tree classifier on private data. In *International Conference on Data Mining Workshop on Privacy, Security and Data Mining (PSDM)*, 2002.

[104] W. Du and Z. Zhan. A practical approach to solve secure multi-party computation problems. In *New security paradigms workshop (NSPW)*, 2002.

[105] T. Duong. Security and privacy in Google Allo, May 2016. `https://vnhacker.blogspot.com/2016/05/security-and-privacy-in-google-allo.html`.

[106] Z. Durumeric, D. Adrian, A. Mirian, J. Kasten, E. Bursztein, N. Lidzborski, K. Thomas, V. Eranti, M. Bailey, and J. A. Halderman. Neither snow nor rain nor MITM...: An empirical analysis of email delivery security. In *ACM SIGCOMM Conference on Internet Measurement (IMC)*, 2015.

[107] Electronic Frontier Foundation. NSA spying on Americans. `https://www.eff.org/nsa-spying`.

[108] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research (JMLR)*, 9(Aug):1871–1874, 2008.

[109] L. Franceschi-Bicchierai. Even the inventor of PGP doesn't use PGP, 2015. `http://motherboard.vice.com/read/even-the-inventor-of-pgp-doesnt-use-pgp`.

[110] W. Gasarch. A survey on private information retrieval. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 82:72–107, 2004.

[111] C. Gentry. Fully homomorphic encryption using ideal lattices. In *ACM Symposium on Theory of Computing (STOC)*, 2009.

[112] C. Gentry, S. Halevi, and N. P. Smart. Homomorphic evaluation of the aes circuit. In *Advances in Cryptology—CRYPTO*. 2012.

[113] Y. Gertner, S. Goldwasser, and T. Malkin. A random server model for private information retrieval. In *International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, 1998.

[114] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. In *ACM Symposium on Theory of Computing (STOC)*, 1998.

[115] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikäinen. On private scalar product computation for privacy-preserving data mining. In *International Conference on Information Security and Cryptology (ICISC)*. 2004.

[116] I. Goldberg. Percy++ project on SourceForge. `http://percy.sourceforge.net/`.

[117] I. Goldberg. Improving the robustness of private information retrieval. In *IEEE Symposium on Security and Privacy (S&P)*, 2007.

[118] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *ACM Symposium on Theory of Computing (STOC)*, 1987.

[119] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[120] J. Goodman and W.-t. Yih. Online discriminative spam filter training. In *Conference on Email and Anti-Spam (CEAS)*, 2006.

[121] Google. Google transparency report. `https://www.google.com/transparencyreport/userdatarequests/US/`.

[122] Google. How Gmail ads work. `https://support.google.com/mail/answer/6603?hl=en`.

[123] Google. Topics used for personalized ads. `https://support.google.com/ads/answer/2842480?hl=en`.

[124] S. D. Gordon, J. Katz, V. Kolesnikov, F. Krell, T. Malkin, M. Raykova, and Y. Vahlis. Secure two-party computation in sublinear (amortized) time. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.

[125] J. Gould. The natural history of Gmail data mining. Gmail isn't really about email—it's a gigantic profiling machine. *Medium*, June 2014. `https://medium.com/@jeffgould/the-natural-history-of-gmail-data-mining-be115d196b10`.

[126] V. Goyal. Multiple description coding: compression meets the network. *IEEE Signal Processing Magazine*, 18(5):74–93, Sept. 2001.

[127] P. Graham. A plan for spam, 2002. `http://www.paulgraham.com/spam.html`.

[128] P. Graham. Better Bayesian filtering, 2003. `http://www.paulgraham.com/better.html`.

[129] S. Guha, B. Cheng, and P. Francis. Privad: Practical privacy in online advertising. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2011.

[130] T. Gupta, N. Crooks, W. Mulhern, S. Setty, L. Alvisi, and M. Walfish. Scalable and private media consumption with Popcorn. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016.

[131] T. Gupta, H. Fingler, L. Alvisi, and M. Walfish. Pretzel: Email encryption and provider-supplied functions are compatible. In *ACM SIGCOMM Conference*, 2017.

[132] S. Gürses, C. Troncoso, and C. Diaz. Engineering Privacy by Design. In *Proc. International Conference on Privacy and Data Protection (CPDP)*, 2011.

[133] R. Henry, Y. Huang, and I. Goldberg. One (block) size fits all: PIR and SPIR over arbitrary-length records via multi-block PIR queries. In *Network and Distributed System Security Symposium (NDSS)*, 2013.

[134] R. Henry, F. Olumofin, and I. Goldberg. Practical PIR for electronic commerce. In *ACM Conference on Computer and Communications Security (CCS)*, 2011.

[135] G. B. Horn, P. Knudsgaard, S. B. Lassen, M. Luby, and J. E. Rasmussen. A scalable and reliable paradigm for media on demand. *IEEE Computer*, 34(9):40–45, 2001.

[136] U. Horn, K. Stuhlmüller, M. Link, and B. Girod. Robust internet video transmission based on scalable coding and unequal error protection. *Signal Processing: Image Communication*, 15(1):77–94, 1999.

[137] J. Huang, J. Lu, and C. X. Ling. Comparing Naive Bayes, decision trees, and SVM with AUC and accuracy. In *IEEE International Conference on Data Mining (ICDM)*, 2003.

[138] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium (SEC)*, 2011.

[139] Y. Huang and I. Goldberg. Outsourced private information retrieval. In *Workshop on Privacy in the Electronic Society (WPES)*, 2013.

[140] Y. Huang, J. Katz, and D. Evans. Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In *IEEE Symposium on Security and Privacy (S&P)*, 2012.

[141] Y. Huang, Z. Lu, et al. Privacy preserving association rule mining with scalar product. In *International Conference on Natural Language Processing and Knowledge Engineering (NLP-KE)*, 2005.

[142] Y. Huang, L. Malka, D. Evans, and J. Katz. Efficient privacy-preserving biometric identification. In *Network and Distributed System Security Symposium (NDSS)*, 2011.

[143] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel. Ryoan: A distributed sandbox for untrusted computation on secret data. In *OSDI*, 2016.

[144] S. Huss-Lederman, E. M. Jacobson, J. Johnson, A. Tsao, and T. Turnbull. Implementation of Strassen's algorithm for matrix multiplication. In *ACM/IEEE Conference on Supercomputing*, 1996.

[145] A. Iliev and S. Smith. Private information storage with logarithmic-space secure hardware. In *Information Security Management, Education and Privacy*, 2004.

[146] I. Ioannidis, A. Grama, and M. Atallah. A secure protocol for computing dot-products in clustered and distributed environments. In *International Conference on Parallel Processing (ICPP)*, 2002.

[147] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology—CRYPTO*. 2003.

[148] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Batch codes and their applications. In *ACM Symposium on Theory of Computing (STOC)*, 2004.

[149] S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Outsourced symmetric private information retrieval. In *ACM Conference on Computer and Communications Security (CCS)*, 2013.

[150] S. Jha, L. Kruger, and P. McDaniel. Privacy preserving clustering. In *European Symposium on Research in Computer Security (ESORICS)*, 2005.

[151] T. Joachims. Text categorization with Support Vector Machines: Learning with many relevant features. In *European Conference on Machine Learning (ECML)*, 1998.

[152] M. Johanson and A. Lie. Layered encoding and transmission of video in heterogeneous environments. In *ACM Multimedia (ACM-MM)*, 2002.

[153] S. Johnson, V. Scarlata, C. Rozas, E. Brickell, and F. Mckeen. Intel software guard extensions: EPID provisioning and attestation services. https://software.intel.com/en-us/blogs/2016/03/09/intel-sgx-epid-provisioning-and-attestation-services,
2016.

[154] J. Joskowicz and J. Ardao. Combining the effects of frame rate, bit rate, display size and video content in a parametric video quality model. In *Latin America Networking Conference (LANC)*, 2011.

[155] C. Kaleli and H. Polat. Providing Naïve Bayesian classifier-based private recommendations on partitioned data. In *European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, 2007.

[156] J.-S. Kang and D. Hong. On fast private scalar product protocols. In *Security Technology (SecTech)*, pages 1–10. Springer, 2011.

[157] S. Katzenbeisser and M. Petković. Privacy-preserving recommendation systems for consumer healthcare services. In *International Conference on Availability, Reliability and Security (ARES)*, 2008.

[158] M. Keller, E. Orsini, and P. Scholl. Actively secure OT extension with optimal overhead. In *Advances in Cryptology—CRYPTO*, 2015.

[159] A. Khedr, G. Gulak, and V. Vaikuntanathan. SHIELD: Scalable homomorphic implementation of encrypted data-classifiers. *IEEE Transactions on Computers*, 65(9):2848–2858, 2014.

[160] B. Kreuter, A. Shelat, and C.-H. Shen. Billion-gate secure computation with malicious adversaries. In *USENIX Security Symposium (SEC)*, pages 285–300, 2012.

[161] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Symposium on Foundations of Computer Science (FOCS)*, 1997.

[162] A. Kwon, D. Lazar, S. Devadas, and B. Ford. Riffle: An efficient communication system with strong anonymity. In *Privacy Enhancing Technologies Symposium (PETS)*, 2016.

[163] M. D. Lam, E. E. Rothberg, and M. E. Wolf. The cache performance and optimizations of blocked algorithms. *ACM SIGOPS Operating Systems Review*, 25(Special Issue):63–74, 1991.

[164] S. Laur and H. Lipmaa. On private similarity search protocols. In *Nordic Workshop on Secure IT Systems (NordSec)*, 2004.

[165] M. Z. Lee, A. M. Dunn, B. Waters, E. Witchel, and J. Katz. Anon-pass: Practical anonymous subscriptions. In *IEEE Symposium on Security and Privacy (S&P)*, 2013.

[166] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research (JMLR)*, 5(Apr):361–397, 2004.

[167] C.-J. Lin, R. C. Weng, and S. S. Keerthi. Trust region Newton method for logistic regression. *Journal of Machine Learning Research (JMLR)*, 9(Apr):627–650, 2008.

[168] Y. Lindell. Fast cut-and-choose-based protocols for malicious and covert adversaries. *Journal of Cryptology*, 29(2):456–490, 2016.

[169] Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[170] X. Liu, R. Lu, J. Ma, L. Chen, and B. Qin. Privacy-preserving patient-centric clinical decision support system on Naive Bayesian classification. *IEEE Journal of Biomedical and Health Informatics*, 20(2):655–668, 2016.

[171] J. R. Lorch, B. Parno, J. Mickens, M. Raykova, and J. Schiffman. Shroud: Ensuring private access to large-scale data in the data center. In *USENIX Conference on File and Storage Technologies (FAST)*, 2013.

[172] W. Lueks and I. Goldberg. Sublinear scaling for multi-client private information retrieval. In *International Conference on Financial Cryptography and Data Security (FC)*, 2015.

[173] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2010.

[174] V. Lyubashevsky, C. Peikert, and O. Regev. A toolkit for ring-LWE cryptography. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2013.

[175] M. Maas, E. Love, E. Stefanov, M. Tiwari, E. Shi, K. Asanovic, J. Kubiatowicz, and D. Song. Phantom: Practical oblivious computation in a secure processor. In *ACM Conference on Computer and Communications Security (CCS)*, 2013.

[176] D. Malkhi, N. Nisan, B. Pinkas, Y. Sella, et al. Fairplay-secure two-party computation system. In *USENIX Security Symposium (SEC)*, 2004.

[177] T. Mayberry, E.-O. Blass, and A. H. Chan. PIRMAP: Efficient private information retrieval for MapReduce. In *International Conference on Financial Cryptography and Data Security (FC)*, 2013.

[178] T. Mayberry, E.-O. Blass, and A. H. Chan. Efficient private file retrieval by combining ORAM and PIR. In *Network and Distributed System Security Symposium (NDSS)*, 2014.

[179] A. McCallum, K. Nigam, et al. A comparison of event models for Naive Bayes text classification. In *AAAI workshop on learning for text categorization*, pages 41–48, 1998.

[180] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman. CONIKS: Bringing key transparency to end users. In *USENIX Security Symposium (SEC)*, 2015.

[181] C. A. Melchor, B. Crespin, P. Gaborit, V. Jolivet, and P. Rousseau. High-speed private information retrieval computation on GPU. In *International Conference on Emerging Security Information, Systems and Technologies (SECUREWARE)*, 2008.

[182] R. C. Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21(4):294–299, Apr. 1978.

[183] V. Metsis, I. Androutsopoulos, and G. Paliouras. Spam filtering with Naive Bayes–which Naive Bayes? In *Conference on Email and Anti-Spam (CEAS)*, 2006.

[184] T. Meyer. No warrant, no problem: How the government can get your digital data. *ProPublica*, June 2014. `https://www.propublica.org/special/no-warrant-no-problem-how-the-government-can-still-get-your-digital-data/`.

[185] Microsoft. Law enforcement requests report. `https://www.microsoft.com/about/csr/transparencyhub/lerr/`.

[186] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *ACM Symposium on Theory of Computing (STOC)*, 1999.

[187] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *IEEE Symposium on Security and Privacy (S&P)*, 2008.

[188] A. Narayanan and V. Shmatikov. Myths and fallacies of "personally identifiable information". *Communications of the ACM*, 53(6):24–26, June 2010.

[189] A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. In *Advances in Neural Information Processing Systems (NIPS)*, 2001.

[190] F. Olumofin and I. Goldberg. Preserving access privacy over large databases. Technical Report 33, University of Waterloo, 2010.

[191] F. Olumofin, P. K. Tysowski, I. Goldberg, and U. Hengartner. Achieving efficient query privacy for location based services. In *Privacy Enhancing Technologies Symposium (PETS)*, 2010.

[192] R. Ostrovsky and W. E. Skeith III. A survey of single-database private information retrieval: Techniques and applications. In *Public Key Cryptography (PKC)*, 2007.

[193] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 1999.

[194] S. Papadopoulos, S. Bakiras, and D. Papadias. pCloud: A distributed system for practical PIR. *IEEE Transactions on Dependable and Secure Computing*, 9(1):115–127, 2012.

[195] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. Keromytis, and S. Bellovin. Blind Seer: A scalable private DBMS. In *IEEE Symposium on Security and Privacy (S&P)*, 2014.

[196] M. A. Pathak, M. Sharifi, and B. Raj. Privacy preserving spam filtering. *arXiv preprint arXiv:1102.4021*, 2011.

[197] C. Peikert. How (not) to instantiate ring-LWE. In *Conference on Security and Cryptography for Networks (SCN)*, 2016.

[198] S. Perez. Microsoft's Cortana can now create reminders from your emails, Feb. 2017. `https://techcrunch.com/2017/02/09/microsofts-cortana-can-now-create-reminders-from-your-emails/`.

[199] Pomelo LLC. Analysis of Netflix's security framework for 'Watch Instantly service. `http://pomelollc.files.wordpress.com/2009/04/pomelo-tech-report-netflix.pdf`, Mar. 2009.

[200] K. Poulsen. Five IRS employees charged with snooping on tax returns. *Wired*, May 2008. `https://www.wired.com/2008/05/five-irs-employ/`.

[201] H. M. Radha, M. Van der Schaar, and Y. Chen. The MPEG-4 fine-grained scalable video coding method for multimedia streaming over IP. *IEEE Transactions on Multimedia*, 3(1):53–68, 2001.

[202] D. Rayburn. CDN market trends: Pricing, growth and competitive landscape. In *Content Delivery Summit*, 2015.

[203] D. Rebollo-Monedero and J. Forné. Optimized query forgery for private information retrieval. *IEEE Transactions on Information Theory*, 56(9):4631–4642, 2010.

[204] A. R. Reibman, H. Jafarkhani, Y. Wang, M. T. Orchard, and R. Puri. Multiple description coding for video using motion compensated prediction. In *International Conference on Image Processing (ICIP)*, 1999.

[205] G. Robinson. A statistical approach to the spam problem. *Linux Journal*, Mar. 2003. `http://www.linuxjournal.com/article/6467`.

[206] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede. Compact ring-LWE cryptoprocessor. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2014.

[207] I. S. Rubinstein. Regulating privacy by design. *Berkeley Technology Law Journal*, 26:1409–1455, 2012.

[208] A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. Efficient privacy-preserving face recognition. In *International Conference on Information Security and Cryptology (ICISC)*, 2009.

[209] A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. Efficient privacy-preserving face recognition (full version). Cryptology ePrint Archive, Report 2009/507, 2009.

[210] F. Saint-Jean. Java implementation of a single-database computationally symmetric private information retrieval (cSPIR) protocol. Technical report, DTIC Document, 2005.

[211] P. Schaar. Privacy by Design. *Identity in the Information Society*, 3(2):267–274, 2010.

[212] B. Schneier. The eternal value of privacy. *Wired*, May 2006. `http://archive.wired.com/politics/security/commentary/securitymatters/2006/05/70886`.

[213] H. Schwarz, D. Marpe, and T. Wiegand. Overview of the scalable video coding extension of the H.264/AVC standard. *Transactions on Circuits and Systems for Video Technology*, 17(9):1103–1120, 2007.

[214] D. Sculley and G. Wachman. Relaxed online SVMs in the TREC spam filtering track. In *Text REtrieval Conference (TREC)*, 2007.

[215] D. Sculley and G. M. Wachman. Relaxed online SVMs for spam filtering. In *ACM SIGIR Conference*, 2007.

[216] R. Shaltiel. Recent developments in explicit constructions of extractors. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 77(67-95):10, 2002.

[217] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[218] M. Shaneck and Y. Kim. Efficient cryptographic primitives for private data mining. In *Hawaii International Conference on System Sciences (HICSS)*, 2010.

[219] R. Singel. Netflix spilled your *Brokeback Mountain* secret, lawsuit claims. *Wired*, Dec. 2009. `http://www.wired.com/images_blogs/threatlevel/2009/12/doe-v-netflix.pdf`.

[220] K. D. Singh, Y. Hadjadj-Aoul, and G. Rubino. Quality of experience estimation for adaptive HTTP/TCP video streaming using H.264/AVC. In *Consumer Communications and Networking Conference (CCNC)*, 2012.

[221] R. Sion and B. Carbunar. On the practicality of private information retrieval. In *Network and Distributed System Security Symposium (NDSS)*, Mar. 2007.

[222] S. W. Smith and D. Safford. Practical server privacy with secure coprocessors. *IBM Systems Journal*, 40(3):683–695, 2001.

[223] C. Soghoian. Two honest Google employees: our products don't protect your privacy, Nov. 2011. `http://paranoia.dubfire.net/2011/11/two-honest-google-employees-our.html`.

[224] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, T. Schneider, and F. Koushanfar. TinyGarble: Highly compressed and scalable sequential garbled circuits. In *IEEE Symposium on Security and Privacy (S&P)*, 2015.

[225] E. Stefanov, E. Shi, and D. Song. Towards practical oblivious RAM. In *Network and Distributed System Security Symposium (NDSS)*, 2012.

[226] J. P. Stern. A new and efficient all-or-nothing disclosure of secrets protocol. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, 1998.

[227] J. Summers, T. Brecht, D. Eager, and B. Wong. Methodologies for generating HTTP streaming video workloads to evaluate web server performance. In *International Systems and Storage Conference (SYSTOR)*, 2012.

[228] B. Tang, S. Kay, and H. He. Toward optimal feature selection in Naive Bayes for text categorization. *IEEE Transactions on Knowledge and Data Engineering*, 28(9):2508–2521, 2016.

[229] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas. Adnostic: Privacy preserving targeted advertising. In *Network and Distributed System Security Symposium (NDSS)*, 2010.

[230] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction APIs. In *USENIX Security Symposium (SEC)*, 2016.

[231] D. Trincă and S. Rajasekaran. Fast cryptographic multi-party protocols for computing boolean scalar products with applications to privacy-preserving association rule mining in vertically partitioned data. In *Data Warehousing and Knowledge Discovery (DaWaK)*. 2007.

[232] J. Trostle and A. Parrish. Efficient computationally private information retrieval from anonymity or trapdoor groups. In *International Security Conference (ISC)*, 2010.

[233] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich. Processing analytical queries over encrypted data. *Proceedings of the VLDB Endowment (PVLDB)*, 6(5):289–300, Mar. 2013.

[234] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2002.

[235] J. Vaidya, M. Kantarcıoğlu, and C. Clifton. Privacy-preserving Naive Bayes classification. *The VLDB Journal*, 17(4):879–898, 2008.

[236] J. Vaidya, B. Shafiq, A. Basu, and Y. Hong. Differentially private Naive Bayes classification. In *IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, 2013.

[237] J. Vaidya, H. Yu, and X. Jiang. Privacy-preserving SVM classification. *Knowledge and Information Systems*, 14(2):161–178, 2008.

[238] J. van den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *ACM Symposium on Operating Systems Principles (SOSP)*, Oct. 2015.

[239] L. Vanderkam. Stop checking your email, now. *Fortune*, Oct. 2012. http://fortune.com/2012/10/08/stop-checking-your-email-now/.

[240] S. Viswanathan and T. Imielinski. Pyramid broadcasting for video-on-demand service. In *Multimedia Computing and Networking (MMCN)*, 1995.

[241] N. Šrndic and P. Laskov. Practical evasion of a learning-based classifier: A case study. In *IEEE Symposium on Security and Privacy (S&P)*, 2014.

[242] S. Wang, D. Agrawal, and A. El Abbadi. Generalizing PIR for practical private retrieval of public data. In *Working Conference on Data and Applications Security and Privacy (DBSec)*, 2010.

[243] Y. Wang and S. Lin. Error-resilient video coding using multiple description motion compensation. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(6):438–452, 2002.

[244] WhatsApp. WhatsApp FAQ - End-to-End Encryption. https://www.whatsapp.com/faq/en/general/28030015.

[245] Wikipedia. 2016 Democratic National Committee email leak, 2014. https://en.wikipedia.org/wiki/2016_Democratic_National_Committee_email_leak.

[246] Wikipedia. Sony pictures entertainment hack, 2014. https://en.wikipedia.org/wiki/Sony_Pictures_Entertainment_hack.

[247] P. Williams and R. Sion. Usable PIR. In *Network and Distributed System Security Symposium (NDSS)*, 2008.

[248] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Scalable anonymous group communication in the anytrust model. In *European Workshop on Systems Security (EuroSec)*, 2012.

[249] R. Wright and Z. Yang. Privacy-preserving Bayesian network structure computation on distributed heterogeneous data. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2004.

[250] Yahoo! Transparency report: Overview. https://transparency.yahoo.com/.

[251] Z. Yang, S. Zhong, and R. N. Wright. Privacy-preserving classification of customer data without loss of accuracy. In *SIAM International Conference on Data Mining (SDM)*, 2005.

[252] A. C. Yao. Protocols for secure computations. In *Symposium on Foundations of Computer Science (SFCS)*, 1982.

[253] S. Yekhanin. Private Information Retrieval. *Communications of the ACM*, 53(4):68–73, Apr. 2010.

[254] X. Yi, M. G. Kaosar, R. Paulet, and E. Bertino. Single-database private information retrieval from fully homomorphic encryption. *IEEE Transactions on Knowledge and Data Engineering*, 25(5):1125–1134, 2013.

[255] X. Yi and Y. Zhang. Privacy-preserving Naive Bayes classification on distributed data via semi-trusted mixers. *Information Systems*, 34(3):371–380, 2009.

[256] H. Yu, X. Jiang, and J. Vaidya. Privacy-preserving SVM using nonlinear kernels on horizontally partitioned data. In *ACM Symposium on Applied Computing (SAC)*, 2006.

[257] H. Yu, J. Vaidya, and X. Jiang. Privacy-preserving SVM classification on vertically partitioned data. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2006.

[258] H. Yu, D. Zheng, B. Y. Zhao, and W. Zheng. Understanding user behavior in large-scale video-on-demand systems. *ACM SIGOPS Operating Systems Review*, 40(4):333–344, 2006.

[259] S. Zahur and D. Evans. Obliv-C: A language for extensible data-oblivious computation. Cryptology ePrint Archive, Report 2015/1153, 2015.

[260] S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. 2015.

[261] S. Zahur, X. Wang, M. Raykova, A. Gascón, J. Doerner, D. Evans, and J. Katz. Revisiting square-root ORAM efficient random access in multi-party computation. In *IEEE Symposium on Security and Privacy (S&P)*, 2016.

[262] K. Zetter. Ex-Googler allegedly spied on user e-mails, chats, Sept. 2010. `https://www.wired.com/2010/09/google-spy/`.

[263] H. Zhang. The optimality of Naive Bayes. *AA*, 1(2):3, 2004.

[264] P. Zhang, Y. Tong, S. Tang, and D. Yang. Privacy preserving Naive Bayes classification. In *Advanced Data Mining and Applications (ADMA)*. 2005.

[265] Y. Zhu, Z. Wang, B. Hassan, Y. Zhang, J. Wang, and C. Qian. Fast secure scalar product protocol with (almost) optimal efficiency. In *Collaborative Computing: Networking, Applications, and Worksharing (CollaborateCom)*. 2015.

[266] P. R. Zimmermann. *The official PGP user's guide*. MIT press, 1995.