Pretzel: Email encryption and provider-supplied functions are compatible

Trinabh Gupta[*†] Henrique Fingler[*] Lorenzo Alvisi[*‡] Michael Walfish[†]

[*]UT Austin [†]NYU [‡]Cornell

ABSTRACT

Emails today are often encrypted, but only between mail servers—the vast majority of emails are exposed in plaintext to the mail servers that handle them. While better than no encryption, this arrangement leaves open the possibility of attacks, privacy violations, and other disclosures. Publicly, email providers have stated that default end-to-end encryption would conflict with essential functions (spam filtering, etc.), because the latter requires analyzing email text. The goal of this paper is to demonstrate that there is no conflict. We do so by designing, implementing, and evaluating *Pretzel*. Starting from a cryptographic protocol that enables two parties to jointly perform a classification task without revealing their inputs to each other, Pretzel refines and adapts this protocol to the email context. Our experimental evaluation of a prototype demonstrates that email can be encrypted end-to-end *and* providers can compute over it, at tolerable cost: clients must devote some storage and processing, and provider overhead is roughly 5× versus the status quo.

CCS CONCEPTS

• **Information systems** → **Email**; • **Security and privacy** → **Cryptography**; **Privacy-preserving protocols**;

KEYWORDS

encrypted email, secure two-party computation, linear classifiers

ACM Reference format:
Trinabh Gupta, Henrique Fingler, Lorenzo Alvisi, and Michael Walfish. 2017. Pretzel: Email encryption and provider-supplied functions are compatible. In *Proceedings of SIGCOMM '17, Los Angeles, CA, USA, August 21-25, 2017,* 14 pages. https://doi.org/10.1145/3098822.3098835

1 INTRODUCTION

Email is ubiquitous and fundamental. For many, it is the principal communication medium, even with intimates. For these reasons, and others that we outline below, our animating ideal in this paper 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 [47, 56]). 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 provider, handle emails in plaintext. Since these providers are typically well-run services with a reputation to protect, many users are willing to just trust them. This trust however, appears to stem more from shifting *social* norms than from the fundamental technical safety of the arrangement, which instead seems to call for greater caution.

Reputable organizations have been known to unwittingly harbor rogue employees bent on gaining access to user email accounts and other private user information [27, 103, 140]. If you were developing your latest startup idea over email, would you be willing to bet its viability on the assumption that each employee within the provider acts properly? And well-run organizations are not immune from hacks [127, 128]—nor from the law. Just in the first half of 2013, Google [64], Microsoft [97] and Yahoo! [131] collectively received over 29,000 requests for email data from law enforcement, and in the vast majority of cases responded with some customer data [96].

End-to-end email encryption can shield email contents from prying eyes and reduce privacy loss when email providers are hacked; and, while authorities would still be able to acquire private email by serving subpoenas to account owners, they would not gain unfettered access to someone's private correspondence without that party's knowledge.

Why then are emails not encrypted end-to-end by default? After all, there has long been software that implements this functionality, notably PGP [144]; moreover, the large webmail providers offer it as an option [63, 130] (see also [23, 113, 115, 126]). A crucial reason—at least the one that is often cited [41, 42, 55, 65, 112]—is that encryption appears to be incompatible with value-added functions (such as spam filtering, email search, and predictive personal assistance [28, 39, 49, 102]) and with the functions by which "free" webmail providers monetize user data (for example, topic extraction) [67]. 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 [117]). So it follows that the functions must execute on providers' servers with access to plaintext emails.

But does that truly follow? Our objective in this paper is to refute these claims of incompatibility, and thus move a step closer to the animating ideal that we stated at the outset, by building an alternative, called Pretzel.

In Pretzel, senders encrypt email using an end-to-end encryption scheme, and the intended recipients decrypt and obtain email contents. Then, the email provider and each recipient engage in a *secure two-party computation* (2PC); the term refers to cryptographic

SIGCOMM '17, August 21-25, 2017, Los Angeles, CA, USA

ACM ISBN 978-1-4503-4653-5/17/08…$15.00
https://doi.org/10.1145/3098822.3098835

protocols that enable one or both parties to learn the output of an agreed-upon function, without revealing the inputs to each other. For example, a provider supplies its spam filter, a user supplies an email, and both parties learn whether the email is spam while protecting the details of the filter and the content of the email.

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). Therefore, in designing Pretzel, we decided to make certain compromises 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.

The paper's central example is classification, which Pretzel applies to both spam filtering and topic extraction (Pretzel also implements elementary keyword search). Pretzel's first step is to compose (a) a relatively efficient 2PC protocol (§3.2) geared to computations that consist mostly of linear operations [19, 30, 75, 100, 106], (b) linear classifiers from machine learning (Naive Bayes, SVMs, logistic regression), which fit this form and have good accuracy (§3.1), and (c) mechanisms that protect against adversarial parties. Although the precise protocol (§3.3) 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, first, with lower-level protocol refinements: revisiting the cryptosystem (§4.1) and conserving calls into it by applying a packing technique [59] (§4.2). Second, for topic extraction, Pretzel rearranges the setup, by decomposing classification into a non-private step, performed by the client, which prunes the set of topics; and a private step that further refines this candidate set to a single topic. Making this work requires a modified protocol that, roughly speaking, selects a candidate maximum from a particular subset, while hiding that subset (§4.3). Third, Pretzel applies well-known ideas (feature selection to reduce costs, various mechanisms to guard against misuses of the protocol); here, the work is demonstrating that these are suitable in the present context.

We freely admit that not all 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.

Indeed, evaluation (§6) of our implementation (§5) indicates that Pretzel's cost, versus a legacy non-private implementation, is estimated to be up to 5.4×, with additional client-side requirements of several hundred megabytes of storage and per-email CPU cost of several hundred milliseconds. These costs represent reductions versus the starting point (§3.3) of between 1.8× and 100× (§6).

Our work here has clear limitations (§7). Reflecting its prototype status, our implementation handles only the three functions mentioned (ideally, it would handle predictive personal assistance, virus scanning, and more); also, Pretzel does not hide metadata, and it focuses on applying classifiers, not training or retraining them. More fundamentally, Pretzel compromises on functionality; by its design, both user and provider have to agree on the algorithm, with only the inputs being private. Most fundamentally, Pretzel cannot achieve the ideal of perfect privacy; it seems inherent in the problem setup that one party gains information that would ideally be hidden. However, these leaks are generally bounded, and concerned users can opt out, possibly at some dollar cost (§4.4, §7).

The biggest limitation, though, is that Pretzel cannot change the world on its own. As we discuss later (§7), there are other obstacles en route to the ultimate goal: general deployment difficulties, key management, usability, and even politics. However, we hope that the exercise of working through the technical details to produce an existence proof (and a rough cost estimate) will at least shape discourse about the viability of default end-to-end email encryption.

2 ARCHITECTURE AND OVERVIEW

2.1 Design ethos: (non)requirements

Pretzel would ideally (a) enable rich computation 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 (§4.4, §7). 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 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

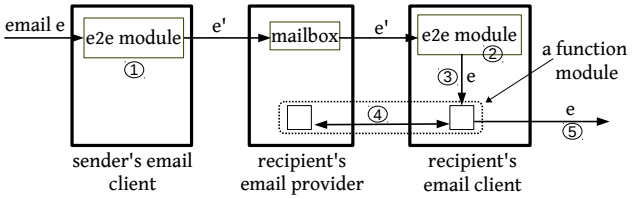

Figure 1: Pretzel's architecture. e denotes plaintext email; e' denotes encrypted email. The sender's provider is not depicted.

are permitted to consume client-side resources, including storage [40]. Furthermore, Pretzel will aim to be configuration-free. Also, Pretzel must be backwards compatible with existing email delivery infrastructure (SMTP, IMAP, etc.).

2.2 Architecture

Figure 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 [1, 37]. 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. Thus, Pretzel requires a solution to *key management* [2, 31, 93, 126]. However, this is a separate effort, deserving of its own paper or product and (as noted in the introduction) is one of the obstacles that Pretzel does not address. Later (§7), we discuss why we are optimistic that it will ultimately be overcome.

The main work for Pretzel surrounds the function modules; the challenge is to balance privacy, functionality, and performance (§2.1). Our focus will be on two modules: spam filtering and topic extraction (§3, §4). We will also report on an elementary keyword search module (§5). 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 BACKGROUND, BASELINE, RELATED WORK

3.1 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) [68, 92, 95, 104] classifiers, specifically a variant of Graham-Robinson's NB [68, 104] for spam filtering (we call this variant GR-NB),[1] and multinomial NB [92] for topic extraction; Logistic Regression (LR) classifiers [57, 62, 86, 98], specifically binary LR [86] and multinomial LR [57] for spam filtering and topic extraction respectively; and linear Support Vector Machine (SVM) classifiers [32, 45, 78, 109], specifically two-class and one-versus-all SVM [32] for spam filtering and topic extraction respectively. These algorithms, or variants of them, yield high accuracy [44, 62, 68, 71, 78, 141] (see also §6.1, §6.2), and are used in popular open-source software packages for spam filtering, classification, and general machine learning [3–7, 57].

The three types of classifiers differ in their underlying assumptions and how they learn parameters from training data. However, when applying a trained 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 (an email, in our context) can belong to one of several *categories* (for example, spam or non-spam). The algorithms output a prediction of a document's category.

Documents 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 [68, 104], 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 $\{(\vec{v}_j,\ p(C_j))\}$ $(1 \le j \le B)$, 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 [8]). $p(C_j)$ 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 [70, Appx A.1] that $\log \alpha$ is equivalent to:

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

where C_1 represents spam and C_2 represents non-spam.

[1] The original Graham-Robinson NB protects against spam emails that hide a short message within a large non-spam text [69]. We do not implement that piece; the resulting change in classification accuracy is small (§6.1).

[2] In more detail, the GR-NB spam classifier assumes that the $\{x_i\}$ are realizations of independent 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.

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 (Gen, Enc, Dec), meaning that $\text{Enc}(pk, m_1) \cdot \text{Enc}(pk, m_2) = \text{Enc}(pk, m_1 + m_2)$, 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 $\text{Enc}(pk, m_1)$, one can compute $\text{Enc}(pk, z \cdot m_1)$.

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 $(pk, sk) \leftarrow \text{Gen}(1^k)$, where k is a security parameter.
   (b) Encrypts each column component-wise, so $\text{Enc}(pk, \vec{v}_j) = (\text{Enc}(pk, v_{1,j}), \ldots, \text{Enc}(pk, v_{N,j}))$.
   (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: $\text{Enc}(pk, d_j) = \text{Enc}(pk, \sum_{i=1}^{N} x_i \cdot v_{i,j})$, 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 $\text{Enc}(pk, d_j + n_j) = \text{Enc}(pk, d_j) \cdot \text{Enc}(pk, n_j)$. Here b is the bit-length of d_j and $\delta \geq 1$ is a security parameter.
   (c) Sends $(\text{Enc}(pk, d_1 + n_1), \ldots, \text{Enc}(pk, d_B + n_B))$ 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 2: Yao+GLLM. This protocol [19, 30, 75, 100, 106] combines GLLM's secure dot products [60] with Yao's general-purpose 2PC [133]. Pretzel's design and implementation apply this protocol to the linear classifiers described in §3.1. The provider is Party X, and the client is Party Y. Pretzel's instantiation of this protocol incorporates several additional elements (§3.3): a variant of Yao [77, 81] 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 (§4.2) in steps 1b and 2a.

For the multinomial NB text classifier, selection works by choosing the category C_{j^*} that maximizes likelihood: $j^* = \text{argmax}_j\, p(C_j \mid \vec{x})$. One can show [70, Appx A.2] that it suffices to select the C_j for which the following is maximal:

$$\left(\sum_{i=1}^{i=N} x_i \cdot \log p(t_i \mid C_j)\right) + 1 \cdot \log p(C_j). \quad (2)$$

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 Secure two-party computation

To perform the computation described above within a function module (§2.2) 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 [61, 133]. Pretzel builds on a relatively efficient 2PC protocol [19, 30, 75, 100, 106] that we name **Yao+GLLM**; we present this below, informally and bottom up (for details and rigorous descriptions, see [60, 72, 88, 107]).

Yao's 2PC. A building block of Yao+GLLM is the classic scheme of Yao [133]. 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 [66, 73, 81, 83, 87, 114, 138, 139] 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 (§6.1, §6.2), despite using a recent and optimized implementation [137, 138].

Secure dot products. Another building block of Yao+GLLM is a secure dot product protocol, specifically GLLM [60]. Many such protocols (also called secure scalar product (SSP) protocols) have

been proposed [21, 25, 51–54, 60, 76, 111, 118, 120, 129, 143]. They fall into two categories: those that are provably secure [51, 60, 129] and those that either have no security proof or require trusting a third party [21, 25, 52–54, 76, 111, 118, 120, 143]. Several protocols in the latter category have been attacked [38, 60, 74, 80, 84]. GLLM [60] 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 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 [100], face recognition using SVM [19], and face and biometric identification using Euclidean distance [30, 75, 106].

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 [33, 89], fully homomorphic encryption [82] (FHE), or a different Yao hybrid [36]. For us, Yao+GLLM appeared to be a more promising starting point. For example, in contrast to the protocol of Khedr et al. [82], 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. [33].[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 [121, 123, 132, 134–136], or a party can release a trained "noisy" model that hides its training data [79, 122, 142]. These works are complementary to Pretzel's focus on applying trained models.

3.3 Baseline protocol

Pretzel begins by applying the Yao+GLLM protocol (Figure 2, §3.2) to the algorithms described in Section 3.1. This works because expressions (1) and (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 (§2.1); 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 [43, 108, 109].

In defining this baseline, we include mechanisms to defend against adversarial parties (§2.1). Specifically, whereas under the classical Yao protocol an actively adversarial party can obtain the other's private inputs [77], Pretzel incorporates a variant [77, 81] 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 honestly, 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.

4 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 $N = 5$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 (§6). Another thing 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 (§4.1–§4.3), the effects of which are summarized in Figure 3; then we describe Pretzel's robustness to misbehaving parties (§4.4).

4.1 Replacing the cryptosystem

Both Pretzel and the baseline require additively homomorphic encryption (Figure 2). The traditional choice for AHE—it is used in prior works [19, 75, 100, 106]—is Paillier [99], 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* [90], a relatively young assumption (which is usually a disadvantage in cryptography) but one that has nonetheless received a lot of recent attention [18, 34, 46, 91, 101, 105]. Specifically, Pretzel incorporates the additively homomorphic cryptosystem of Brakerski and Vaikuntanathan [34], as implemented and optimized by Melchor et al. [20] 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 (§6), and similarly with Enc. The gain is reflected in the cost model (Figure 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 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] For the data point at $N = 70$, $B = 24$ (these variables are defined in Figure 2), Bost et al. report network transfers and computation times (for the two parties) of 1911 KB, 1664 ms, and 1652 ms [33], whereas these overheads are 156.1 KB, 757.9 ms, and 8.6 ms for our implementation of Yao+GLLM (§5) 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.

[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 [48, 50, 94, 110].

	Non-private	Baseline (§3.3)	Pretzel (§4.1–§4.3)
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 (§3.3)
B = number of categories in the model (§3.3)
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. 2)
$b = \log L + b_{\text{in}} + f_{\text{in}}$ (§4.2)
f_{in} = # of bits for the frequency of a feature in an email (§4.2)
$\beta'_{\text{xpir}} := \lfloor B/p_{\text{xpir}} \rfloor + 1/\lfloor p_{\text{xpir}}/k \rfloor$ 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 (§3.3)
p = # of b-bit probabilities packed in a ciphertext (§4.2)
$K_{\text{cpu}}, K_{\text{net}}$ = constants for CPU and network costs (§3.3)
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)
b_{in} = # of bits to encode a model parameter (§4.2)
N' = # of features selected after aggressive feature selection (§4.3) ($N' = N$ if spam)
B' = # of candidate topics ($\ll B$) (§4.3) ($B' = B$ if spam)
s = "left-shift" CPU time in XPIR-BV (§4.2)

Figure 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 3.3. Microbenchmarks are given in §6.

4.2 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 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 [60], Pretzel's baseline (§3.3), and the works that build on GLLM [19, 75]—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 [119] and homomorphic evaluation of AES-128 encryption [59], address the limitation described above, by packing across both columns and rows. These techniques traverse the matrix in row-major order without restricting the packing to be within a row (see the rightmost matrix in Figure 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 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 4. For the sets with p vectors, it packs together all p elements of a row [60]. 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 [59, 119].

Then, to compute dot products (in step 2a in Figure 2) for all columns except those in the rightmost matrix (Figure 4), Pretzel uses the fact that that the elements that need to be added are aligned [60]. 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 $c_1 = \text{Enc}(pk, v_{1,1} \| \ldots \| v_{1,p})$ and $c_2 = \text{Enc}(pk, v_{2,1} \| \ldots \| v_{2,p})$ yields $c_3 = c_1 \cdot c_2 = \text{Enc}(pk, v_{1,1} + v_{2,1} \| \ldots \| v_{1,p} + v_{2,p})$. For this to work, the individual sums (for example, $v_{1,p} + v_{2,p}$) cannot overflow b bits.

For the columns that are in the rightmost matrix (Figure 4), Pretzel performs dot products by exploiting the homomorphism to cyclically rotate the packed elements in a ciphertext [59]. For example, assume $c = \text{Enc}(pk,\ v_{1,1} \| \ldots \| v_{1,k} \| v_{2,1} \| \ldots \| v_{2,k})$ 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 $c' = \text{Enc}(pk, v_{2,1} \| \ldots \| v_{2,k} \| \ldots)$; this is done by applying the "constant multiplication" operation (Figure 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),

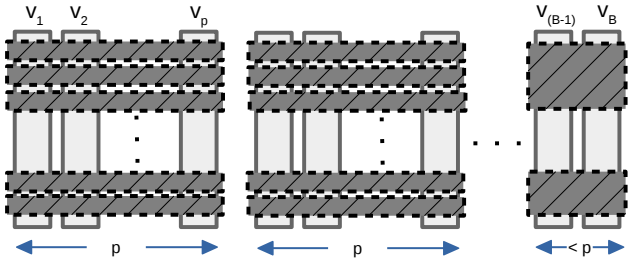

Figure 4: Packing in Pretzel. Light gray rectangles represent matrix columns ($\vec{v}_1, \ldots, \vec{v}_B$); dark gray represent ciphertexts. The arrangement in matrices with p columns follows GLLM [60]; the matrix with $< p$ columns follows Gentry et al. [59].

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 6. For the spam filtering module, the provider's CPU drops by 5× and the client-side storage drops by 7×, relative to the baseline (§3.3). However, CPU at the client increases by 10× (owing to the cyclic shifts), and the network overhead increases by 5.4×; despite these increases, both costs are not exorbitant in absolute terms, and we view them as tolerable (§6.1, §6.2). 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 (§3.3) are smaller: provider CPU drops by 1.37×, client CPU drops by 3.25×, 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.

4.3 Pruning in topic extraction

Decomposed classification. So far, many of the costs are proportional to B: CPU and network cost of Yao (Figure 2, step 4), and storage (Figure 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, 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 [8]. 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.

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 (§6.2).

Finally, step (ii) must not reveal S' to the provider, since that would be more information than a single extracted topic. This rules out instantiating step (ii) by naively applying the existing protocol (§3.3–§4.2), with S' in place of S. Pretzel's response is depicted in Figure 5. There are some low-level details to handle because of the interaction with packing (§4.2), 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 2, "setup phase"). Pretzel's response is the standard technique of *feature selection* [116]: 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). Of course, one presumes that providers already prune their models; the proposal here is to do so more aggressively. Section 6.2 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 §4.2, the provider CPU drops by 45×, client CPU drops by 8.4×, and the network transfers drop by 20.4× (§6.2). Thus, the aforementioned two orders of magnitude (above the non-private version) becomes roughly 5×.

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 2 but with the addition of packing from §4.2.

Computation phase

(3) Party Y does the following:

(a) (compute dot products) As described in Figure 2, step 2a, and §4.2. 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 (§4.2).

(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 2 and §4.2).

(c) (blinding) Blinds $d_{S'[j]}$ using the technique described in Figure 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 5: Protocol for proprietary topic extraction, based on candidate topics (this instantiates step (ii) in Section 4.3). The provider is Party X; the client is Party Y. This protocol builds on the protocol presented in §3.3–§4.2.

4.4 Robustness to misbehaving parties

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

(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 (§3.3). 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 [29, 125]. 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), instead garble an arbitrary function. Similarly, a client could input bogus candidate topics in step (ii) of decomposed classification (§4.3). 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.

5 IMPLEMENTATION

Our prototype fully implements the design described in Section 4. 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 [9] for OpenPGP encryption, Obliv-C [137] for Yao's 2PC protocol,[5] XPIR [20] for the XPIR-BV AHE scheme, LIBLINEAR [24, 57] to train LR and SVM classifiers, and SQLite FTS4 [10] for the search index.

6 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 (§4) help in reducing the overheads?
(3) How accurate are Pretzel's functions: 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 (§3.3). (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 (§3.3).
- 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 (§3.3).
- For topic extraction, the potential coarsening effects of Pretzel's classifiers (§4.3) are a drop in accuracy of between 1–3%.

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

[5] Another choice would have been TinyGarble [114]. 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.

For spam filtering and topic extraction, we compare Pretzel to its starting baseline, which we call **Baseline** (this baseline is described in detail in Section 3.3 and Figure 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 (§5).

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 [22] (481 spam and 2,411 non-spam emails), Enron [11] (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 [12] (18,846 Usenet posts on 20 topics), Reuters-21578 [13] (12,603 newswire stories on 90 topics), and RCV1 [85] (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 [4] and scikit-learn [6].

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 [35]; 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 one `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 (§2.2). Similarly, the client uses a single CPU.

Microbenchmarks. Figure 6 shows the CPU and network costs for the common operations (Figure 3) in Pretzel and the baselines. We will use these microbenchmarks to explain the performance evaluation in the next subsections.

6.1 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 [15–17]. 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.

	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 6: Microbenchmarks for operations shared by Pretzel and the baselines (Figure 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 [14]. Similarly, Yao costs for ϕ = argmax depend linearly on the number of input values; we show costs per input value.

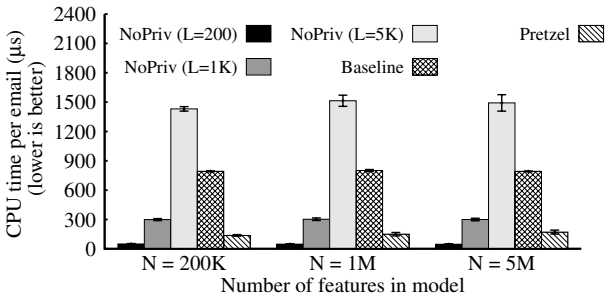

Figure 7: Provider-side CPU time per email in microseconds for the spam filtering module while varying the number of features (N) in the spam classification model, and the number of features (L) in an email. CPU time for NoPriv varies only slightly with N (not visible), while (provider-side) CPU times for Baseline and Pretzel are independent of both L and N (Figure 3).

Provider-side CPU time. Figure 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× Baseline's. Pretzel's is more than NoPriv's because in NoPriv the provider does L feature extractions, map lookups, and float additions, which are fast operations (Figure 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 and §4.1). Pretzel's CPU time is lower than Baseline's because in Pretzel, the provider decrypts a XPIR-BV ciphertext whereas in Baseline the provider decrypts a Paillier ciphertext (Figure 6).

As the number of features in an email increases ($L = \{1000, 5000\}$), the provider's CPU time in both Pretzel and Baseline 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). 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× 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).

	Size		
	N = 200K	N = 1M	N = 5M
Non-encrypted	4.3 MB	21.5 MB	107.3 MB
Baseline	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 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 (§4.2).

	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 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 (§3.1). Also shown is accuracy for the original Graham-Robinson Naive Bayes algorithm (GR).

Client-side overheads. Figure 8 shows the size of the spam model for the various systems. We notice that the model in Pretzel is approximately 7× smaller than the model in Baseline. This is due to the difference in packing in the two systems: "across rows and columns" (in Pretzel) versus "across columns" (in GLLM [60], implemented in Baseline (§4.2). We also notice that, given the refinement of replacing the cryptosystem (§4.1), 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 ($N = 5$M). This time is dominated by the L left shift and add operations in the secure dot product computation (§4.2). Our microbenchmarks (Figure 6) explain this number: 5000 of the left shift and add operation takes $5000 \times 70\mu s = 350$ms. A large L is an unfavorable scenario for Pretzel: client-side processing is proportional to L (Figure 3).

Network transfers. Both Pretzel and Baseline 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 [14]). These overheads are due to transfer of a ciphertext and a comparison inside Yao's framework (Figure 2). Pretzel's overheads are higher than Baseline's because the XPIR-BV ciphertext in Pretzel is much larger than the Paillier ciphertext.

Accuracy. Figure 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

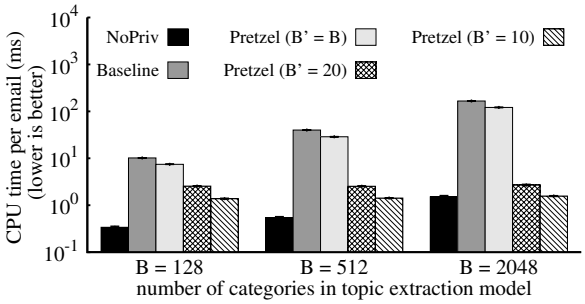

Figure 10: Provider-side CPU time per email in milliseconds for topic extraction, varying the number of categories (B) in the model and the number of candidate topics (B'). The case $B = B'$ measures Pretzel without the decomposed classification technique (§4.3). The y-axis is log-scaled. N and L are 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 Baseline; they increase linearly with L and vary slightly with N for NoPriv.

	network transfers		
	$B = 128$	$B = 512$	$B = 2048$
Baseline	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 11: Network transfers per email for topic extraction in Pretzel and Baseline. B' is the number of candidate topics in decomposed classification (§4.3). Network transfers are independent of the number of features in the model (N) and email (L) (Figure 3).

false positives, or non-spam falsely classified as spam; higher recall means lower false negatives, or spam falsely classified as non-spam.)

6.2 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 $N = \{20K, 100K\}$[6] 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 [8]). For the number of candidate topics for Pretzel (§4.3), we experiment with $B' = \{5, 10, 20, 40\}$.

Provider-side CPU time. Figure 10 shows the per email CPU time consumed by the provider. Without decomposed classification (§4.3)—this is the $B' = B$ case in the figure—Pretzel's CPU time is significantly higher than NoPriv's but lower than Baseline's. Pretzel's time differs from Baseline'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 (§4.3). For $B = 2048, B' = 20$, Pretzel's CPU time is 1.78× NoPriv's; for $B = 2048, B' = 10$, it is 1.03× NoPriv's.

[6]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	
	$N = 20K$	$N = 100K$
Non-encrypted	144.3 MB	769.4 MB
Baseline	288.4 MB	1.5 GB
Pretzel	720.7 MB	3.8 GB

Figure 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 13: Impact of decomposed classification (§4.3) 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.

Network transfers. Figure 11 shows the network transfers per email for Baseline and Pretzel. As expected, with decomposed classification, Pretzel's network transfers are lower; they are 402 KB per email (or 5.4× the average email size of 75 KB, as reported in [14]) 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 12 shows the model sizes (before feature selection; §4.3) for the various systems for different values of N and $B = 2048$. Pretzel's model is bigger than Baseline'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 ($L = 5000$).

Loss of accuracy. Recall that classification accuracy for topic extraction in Pretzel could be affected by decomposed classification and feature selection (§4.3). Figure 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 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 [116]). 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 12.)

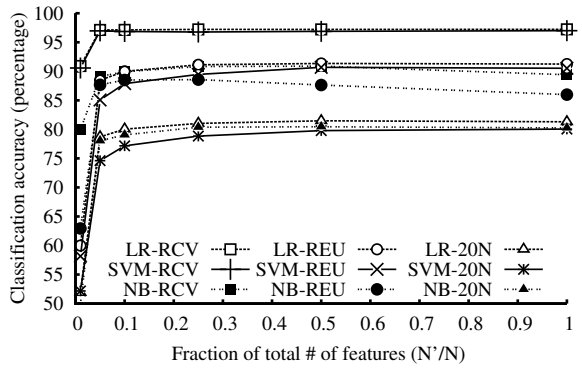

Figure 14: Classification accuracy of topic extraction classifiers in Pretzel as a function of N'/N, where N is the total number of features in the training part of the datasets and N' is the number of selected features (§4.3). The plotted accuracies are for the 20News (20N), Reuters (REU), and RCV1 (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.

	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 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.

6.3 Keyword search and absolute costs

Figure 15 shows the client-side storage and CPU costs of Pretzel's keyword search module (§5).

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 [124] of average size (75 KB) [14], 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.

7 DISCUSSION, LIMITATIONS, FUTURE WORK

Pretzel is an improvement over its baseline (§3.3) of up to 100×, depending on the resource (§6). Its absolute overheads are substantial but, as just discussed (§6.3), 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 [26] is future work. So is adapting Pretzel to hide metadata.

A fundamental limitation of Pretzel is information leakage (§2.1). Section 4.4 discussed this issue and potential remedies. 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 [125]. 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 [121, 123, 132, 134–136]; 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 [58]. On the other hand, there is reason for hope: TLS between data centers was deployed over just several years [56]. 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 [2, 31, 93, 126]. Finally, politics: there are entrenched interests who would prefer email not to be encrypted.

Ultimately, our goal is just to demonstrate an alternative. We don't claim that Pretzel is an optimal point in the three-way tradeoff among functionality, performance, and privacy (§2.1); we don't yet know what such an optimum would be. We simply claim that it is different from the status quo (which combines rich functionality, superb performance, but no encryption by default) and that it is potentially plausible.

Acknowledgments

This draft was improved by comments from and conversations with Varun Chandrasekaran, Eric Crockett, Natacha Crooks, Peter Druschel, Ray Mooney, Ashay Rane, Shabsi Walfish, Shane Williams, Yuanzhong Xu, Samee Zahur, and the anonymous NSDI17 reviewers. We thank Andrew J. Blumberg for an inspiring conversation about webmail (non)privacy. This work was supported by an Amazon EC2 student grant; NSF grants 1055057, 1409555, 1423249, and 1514422; ONR grant N00014-14-1-0469; AFOSR grant FA9550-15-1-0302; and a Google Research Fellowship.

REFERENCES

[1] http://openpgp.org/.
[2] https://keybase.io.
[3] http://spamprobe.sourceforge.net/.
[4] http://spambayes.sourceforge.net/.
[5] http://spamassassin.apache.org/.
[6] http://scikit-learn.org/stable/.
[7] http://www.cs.waikato.ac.nz/ml/weka/.
[8] https://support.google.com/ads/answer/2842480?hl=en.
[9] https://www.gnupg.org/software/gpgme/index.html.
[10] https://www.sqlite.org/fts3.html.
[11] https://www.cs.cmu.edu/~./enron/.
[12] http://qwone.com/~jason/20Newsgroups/.
[13] http://www.daviddlewis.com/resources/testcollections/reuters21578/.
[14] http://email.about.com/od/emailstatistics/f/What_is_the_Average_Size_of_an_Email_Message.htm.
[15] http://www.gossamer-threads.com/lists/spamassassin/users/151578.
[16] http://users.spamassassin.apache.narkive.com/d6ppUDfw/large-scale-global-bayes-tuning.
[17] http://spamassassin.apache.org/full/3.4.x/doc/Mail_SpamAssassin_Conf.html.
[18] A survey on ring-LWE cryptography, Feb. 2016. https://www.microsoft.com/en-us/research/video/a-survey-on-ring-lwe-cryptography/.
[19] 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 *MobiSys*, 2016.
[20] C. Aguilar-Melchor, J. Barrier, L. Fousse, and M.-O. Killijian. XPIR: Private Information Retrieval for Everyone. In *PETS*, 2016.
[21] A. Amirbekyan and V. Estivill-Castro. A new efficient privacy-preserving scalar product protocol. In *Australasian conference on Data mining and analytics (AusDM)*, 2007.
[22] 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.
[23] Apple. Our Approach to Privacy. http://www.apple.com/privacy/approach-to-privacy/.
[24] M. L. G. at National Taiwan University. LIBLINEAR–A library for large linear classification. https://www.csie.ntu.edu.tw/~cjlin/liblinear/.
[25] M. J. Atallah and W. Du. Secure multi-party computational geometry. In *Workshop on Algorithms and Data Structures (WADS)*. 2001.
[26] M. Ball, T. Malkin, and M. Rosulek. Garbling gadgets for boolean and arithmetic circuits. In *ACM CCS*, 2016.
[27] 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.
[28] 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.
[29] 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 *ECML-PKDD*, 2013.
[30] M. Blanton and P. Gasti. Secure and efficient protocols for iris and fingerprint identification. In *ESORICS*, 2011.
[31] J. Bonneau. EthIKS: Using Ethereum to audit a CONIKS key transparency log. In *FC*, 2016.
[32] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Wkshp on Computational Learning Theory (COLT)*, 1992.
[33] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. In *NDSS*, 2014.
[34] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *CRYPTO*, 2011.
[35] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees*. CRC press, 1984.
[36] 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.
[37] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. OpenPGP message format. RFC 4880, IETF, 2007.
[38] Y.-T. Chiang, D.-W. Wang, C.-J. Liau, and T.-s. Hsu. Secrecy of two-party secure computation. In *IFIP DBSec*. 2005.
[39] P. Ciano. How to use Google Now, Mar. 2014. https://paulciano.org/2014/03/getting-google-now/.
[40] M. Cohen. Web storage overview. https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/.
[41] 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/.
[42] 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/.
[43] J. Corbet. The grumpy editor's guide to bayesian spam filters, 2006. https://lwn.net/Articles/172491/.
[44] G. V. Cormack. TREC 2007 spam track overview. In *TREC*, 2007.
[45] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
[46] 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.
[47] T. Dierks and E. Rescorla. The transport layer security (TLS) protocol version 1.2. RFC 5246, Network Working Group, 2008.
[48] W. Diffie and M. Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.
[49] 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.
[50] Y. Dodis, R. Gennaro, J. Håstad, H. Krawczyk, and T. Rabin. Randomness extraction and key derivation using the CBC, cascade and HMAC modes. In *CRYPTO*, 2004.
[51] C. Dong and L. Chen. A fast secure dot product protocol with application to privacy preserving association rule mining. In *PAKDD*, 2014.
[52] W. Du and M. J. Atallah. Protocols for secure remote database access with approximate matching. In *E-Commerce Security and Privacy*, 2001.
[53] W. Du and Z. Zhan. Building decision tree classifier on private data. In *Intl. Conf. on Data Mining Wkshp on Privacy, Security and Data Mining (PSDM)*, 2002.
[54] W. Du and Z. Zhan. A practical approach to solve secure multi-party computation problems. In *New security paradigms workshop (NSPW)*, 2002.
[55] T. Duong. Security and privacy in Google Allo, May 2016. https://vnhacker.blogspot.com/2016/05/security-and-privacy-in-google-allo.html.
[56] 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 *IMC*, 2015.
[57] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *JMLR*, 9(Aug):1871–1874, 2008.
[58] 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.
[59] C. Gentry, S. Halevi, and N. P. Smart. Homomorphic evaluation of the AES circuit. In *CRYPTO*. 2012.
[60] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikäinen. On private scalar product computation for privacy-preserving data mining. In *Intl. Conf. on Information Security and Cryptology (ICISC)*. 2004.
[61] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*, 1987.
[62] J. Goodman and W.-t. Yih. Online discriminative spam filter training. In *Conf. on Email and Anti-Spam (CEAS)*, 2006.
[63] Google. https://github.com/google/end-to-end.
[64] Google. Google transparency report. https://www.google.com/transparencyreport/userdatarequests/US/.
[65] Google. How Gmail ads work. https://support.google.com/mail/answer/6603?hl=en.
[66] 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 CCS*, 2012.
[67] 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.
[68] P. Graham. A plan for spam, 2002. http://www.paulgraham.com/spam.html.
[69] P. Graham. Better Bayesian filtering, 2003. http://www.paulgraham.com/better.html.
[70] T. Gupta, H. Fingler, L. Alvisi, and M. Walfish. Pretzel: Email encryption and provider-supplied functions are compatible (extended version). *arXiv preprint arXiv:1612.04265*, 2016.
[71] J. Huang, J. Lu, and C. X. Ling. Comparing Naive Bayes, decision trees, and SVM with AUC and accuracy. In *Intl. Conf. on Data Mining (ICDM)*, 2003.
[72] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security*, 2011.
[73] Y. Huang, J. Katz, and D. Evans. Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In *IEEE S&P*, 2012.
[74] 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.
[75] Y. Huang, L. Malka, D. Evans, and J. Katz. Efficient privacy-preserving biometric identification. In *NDSS*, 2011.

[76] 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.

[77] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *CRYPTO*, 2003.

[78] T. Joachims. Text categorization with Support Vector Machines: Learning with many relevant features. In *ECML*, 1998.

[79] C. Kaleli and H. Polat. Providing Naïve Bayesian classifier-based private recommendations on partitioned data. In *PKDD*, 2007.

[80] J.-S. Kang and D. Hong. On fast private scalar product protocols. In *Security Technology (SecTech)*. 2011.

[81] M. Keller, E. Orsini, and P. Scholl. Actively secure OT extension with optimal overhead. In *CRYPTO*, 2015.

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

[83] B. Kreuter, A. Shelat, and C.-H. Shen. Billion-gate secure computation with malicious adversaries. In *USENIX Security*, 2012.

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

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

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

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

[88] 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.

[89] 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.

[90] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, 2010.

[91] V. Lyubashevsky, C. Peikert, and O. Regev. A toolkit for ring-LWE cryptography. In *EUROCRYPT*, 2013.

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

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

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

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

[96] 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/.

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

[98] A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. In *NIPS*, 2001.

[99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, 1999.

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

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

[102] 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/.

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

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

[105] 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.

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

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

[108] D. Sculley and G. Wachman. Relaxed online SVMs in the TREC spam filtering track. In *TREC*, 2007.

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

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

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

[112] 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.

[113] S. Somogyi. Making end-to-end encryption easier to use. *Google Security Blog*, June 2014. https://security.googleblog.com/2014/06/making-end-to-end-encryption-easier-to.html.

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

[115] A. Stamos. User-focused security: End-to-end encryption extension for Yahoo Mail. *Yahoo Tumblr Blog*, Mar. 2015. https://yahoo.tumblr.com/post/113708033335/user-focused-security-end-to-end-encryption.

[116] 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.

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

[118] 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.

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

[120] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *KDD*, 2002.

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

[122] 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.

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

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

[125] N. Šrndic and P. Laskov. Practical evasion of a learning-based classifier: A case study. In *IEEE S&P*, 2014.

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

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

[128] Wikipedia. Sony pictures hack, 2014. https://en.wikipedia.org/wiki/Sony_Pictures_hack.

[129] R. Wright and Z. Yang. Privacy-preserving Bayesian network structure computation on distributed heterogeneous data. In *KDD*, 2004.

[130] Yahoo! https://github.com/yahoo/end-to-end.

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

[132] 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.

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

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

[135] 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.

[136] H. Yu, J. Vaidya, and X. Jiang. Privacy-preserving SVM classification on vertically partitioned data. In *PAKDD*, 2006.

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

[138] S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole. In *EUROCRYPT*. 2015.

[139] 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 S&P*, 2016.

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

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

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

[143] 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.

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