Research statement
Trinabh Gupta, trinabh@cs.utexas.edu

Area. My research interests are in the area of computer systems, in particular, in building robust systems that provide strong security, privacy, and availability properties to users. Some examples of the systems that I have built are an email service that protects email contents from leaks and unauthorized use by the email service provider, an on-demand video delivery service that protects users’ media diet from accidental disclosures by video service provider, and a failure detector that allows distributed systems to start recovery from failures quickly.

A particular interest is systems that protect privacy; I want to help create a world in which people’s right to privacy on the web is not left solely in the hands of laws, policies, trust, and hope, but vigorously defended through technical solutions.

System characteristics. The systems that I build have the following characteristics:

- **Strong theoretical foundations.** I start my design work by identifying theoretical constructs from the literature that are accompanied by formal proofs and provide the required security, privacy, or availability properties. For example, to hide users’ media diet from the video service providers, Popcorn [2] builds upon private information retrieval (PIR), cryptographic protocols that hide the content of client requests provably.

- **Tailored for practical performance.** Theoretical constructs like cryptographic protocols, if applied as they appear in the literature, result in impractical systems with huge resource costs. To bring practicality, I carefully tailor them to the application. For example, the privacy-preserving email service Pretzel [3] extends the underlying cryptographic protocol with a new packing technique to reduce storage space costs.

- **Backed by solid implementations and careful evaluations.** After finishing a paper design for a system, I implement and evaluate a solid prototype. For example, the prototype in [2] is over 10,000 lines of code, serves a video library of a few tens of terabytes, and runs over a hundred machines on Amazon EC2. In addition, the evaluation of the prototype measures fine-grained resource consumption, compares the consumption to several baseline systems, explains the consumption using estimates predicted by mathematical models of the system, and reports the cost (in dollar terms) of serving a movie.

- **Myth-busting.** It is a common perception that to bake strong privacy properties into a system, one has to either make serious sacrifices in the system’s functionality or incur high resource costs. Similarly, it is a common perception that a failure detector that is simultaneously accurate and timely is hard to realize. The systems that I have built refute these suppositions.

- **Interdisciplinary.** While designing my systems I often have to dig into different areas: machine learning, cryptography, etc.

Dissertation work
My dissertation covers two privacy-preserving systems: one for on-demand media and the other for email.
Popcorn [2] investigates whether it is possible to build a Netflix-like media delivery system that hides the content of user requests from the content distributor, respects copyrights, and costs a small multiple of a non-private system (in terms of dollars spent to serve a movie).

Popcorn uses PIR cryptographic protocols; these protocols allow a client to retrieve an object from a server’s library without revealing to the server which object is retrieved. Popcorn addresses several challenges of using PIR for media delivery, including PIR’s high resource consumption: to service each request, the server processes its entire library (because otherwise it can infer which object a user is not interested in). Popcorn amortizes the server-side overhead over multiple requests from different users by splitting the library into exponentially-increasing slices (each slice contains a piece from every movie), configuring the first slice to be “narrow” to keep playback delay small, applying PIR independently to each slice, and processing requests to the slices in batches. Because of these techniques (and others), the cost of serving a movie in Popcorn is 3.87 times that in a non-private system; in contrast, the factor would be 265 if PIR were applied as given in the literature.

Pretzel [3]. Today, email service providers don’t enable end-to-end encrypted email by default because they think it would undermine their ability to implement proprietary value-added functions that require analyzing email text (e.g., spam filtering, topic extraction for targeted advertisements, etc.). Pretzel demonstrates that there is no conflict.

Pretzel starts with a secure two-party computation (2PC) cryptographic protocol that enables an email service provider and a client to jointly run a classification function such as spam filtering and topic extraction without revealing their private inputs—machine learning models and email contents—to each other. Pretzel addresses two issues with a naive instantiation of this protocol: high resource consumption and leakage of private data in the case that one or both parties deviate from the protocol. For example, to reduce resource consumption, Pretzel replaces the underlying encryption scheme with a more modern and efficient one, adds a new efficient packing technique to conserve invocations of the encryption algorithm, and prunes the inputs to the protocol. Pretzel’s overheads are acceptable. For example, for spam filtering, the provider’s per-email CPU consumption is roughly three times that in a non-private system (in contrast, the factor would be 60 if Pretzel’s starting 2PC protocol were applied as given in the literature).

Other projects

Bolt [4, 5] is a cloud-based storage system designed to support an emerging class of “smart home” applications that operate on data from multiple homes. More precisely, Bolt exposes an API to store, query, and share time-series data, and to protect confidentiality and integrity of data from untrusted cloud storage providers. For efficiency, Bolt amortizes the resource costs of encryption, integrity checks, and the cloud’s put/get API calls over multiple proximate data records (e.g., data generated in a time window). Bolt’s performance and resource consumption is significantly better than that of OpenTSDB, a data store designed specifically for time-series data. Bolt is open-sourced and used by several universities for IoT (Internet of Things) related research and experiments.

Failure detection in distributed systems. Web services are typically hosted on data centers with thousands of unreliable components (hosts, network links, etc.). For availability, these services must quickly handle component failures.

Albatross [7] is a failure reporting service that detects failures quickly and reliably (i.e., without false positives). For quickness, Albatross uses inside information—failure information present in the various layers of a system (e.g., whether a process is present in the OS’s
process table)—instead of the traditional long end-to-end timeouts. To eliminate false positives, Albatross disconnects processes that it suspects to have failed, by installing packet dropping rules in a SDN-like network.

Unlike Albatross, Pigeon [6] embraces uncertainty. More specifically, it proposes a new expanded failure detection interface that reports four failure conditions (instead of just “up” and “down”) containing an explicit bit on whether the report is reliable or not. The expanded interface helps applications choose shorter recovery paths.

**Future research directions**

**Privacy-preserving systems.** The increasing number of data leak incidents on the web are making it imperative that we develop and deploy privacy-preserving variants of common online services. Continuing in the line of my dissertation work, I want to investigate how we can build a search engine that hides queries from the service provider, returns personalized results, and has practical performance. One potential approach to building such a system would be to combine PIR and 2PC: use a variant of PIR called “PIR by keywords” to privately select all documents matching a query, and use 2PC to filter documents that are personalized to the user. The challenge, of course, would be to address the high resource consumption of PIR and 2PC protocols.

Another question I want to investigate is how we can build a privacy-preserving recommendation system for e-commerce that hides users’ buying histories.

**Tools to formally prove privacy properties of implementations.** Even if we design our systems to provide strong privacy properties, there is risk of information leak because of implementation bugs. To make systems truly private in practice, I want to investigate how we can formally certify implementations, i.e., prove using machine-checkable proofs that an implementation meets a high-level specification of the required privacy properties.

Recent tools to formally certify software have taken significant strides, but still have several limitations. They do not readily work on existing codebases nor fully automate the process of generating proofs, and they require significant developer effort to annotate code and write specifications of the code’s behavior; etc.

I want to take this research forward and build a tool that works on existing codebases and with very high-level specifications, for example, the server must not know which movie a user is watching. I want to develop this tool by: (1) applying existing tools to Popcorn and Pretzel to understand the tools’ limitations in detail, and (2) devising alternate methods for certification because the limitations mentioned above appear to be fundamental to the method being used.

**Tools to automatically convert non-private code to private code.** Another line of research that I want to pursue is developing toolchains that take as input an application’s codebase and a specification of a privacy property, and output a piece of code that implements the same functionality as the original application but while meeting the privacy property. I am not aware of prior work looking at this automated conversion, but I suspect that the conversion process will involve steps like identifying and writing formal specifications of the behavior of different cryptographic protocols, automatically “matching” specifications to the requirements of the application, proving that two given pieces of code are functionally equivalent [1], etc.
References


