

Research Statement

Raluca Ada Popa

I am interested in building secure systems. Leakage of confidential data plagues many computing systems today and my Ph.D. work has focused on protecting data confidentiality against a powerful and common class of attackers: *attackers with full access to data stored on servers*. Such attackers occur in numerous settings. First, due to the recent shift towards cloud computing, an increasing number of companies and users host data at external cloud providers, so their sensitive data becomes readily available to cloud administrators. Indeed, recent surveys show that security remains one of the top concerns for customers of cloud computing and an impediment to adopting clouds for potential customers. Second, hackers notoriously break into systems and gain access to a part or to all of the data stored on a server. Third, we know from recent news that the government has been accessing a lot of private data even without a subpoena. Finally, even inside a local data center, insider attackers are responsible for many data leaks.

Most existing systems today do not protect confidentiality against these attackers. For systems that need to compute on sensitive data, the common approach has been to assume that a part of the server is trusted, either uncompromisable or inaccessible to the adversary, such as the database or the operating system; then, they prevent attacks to the application by using techniques such as static or dynamic analysis, language-based enforcement, or information flow control. Unfortunately, this assumption does not hold for the attackers we described, because they can have access to all server components and can simply read the data from the trusted component (e.g., from the database or main memory).

My work provides confidentiality guarantees against this powerful and relevant class of attackers.

My approach. I take a different approach from these previous works: one should not trust any component at the server; instead, the server should store only encrypted data and be able to process and compute on it efficiently without receiving the decryption key. In this way, even if an attacker gets access to all the data stored at the server, the attacker sees only encrypted data.

The cryptographic community has already proposed computing on encrypted data.¹ However, the proposed schemes, such as fully homomorphic encryption (FHE), are prohibitively impractical for most systems: they are currently orders of magnitude slower than unencrypted computation. A significant challenge lies in building *practical systems*.

I have designed and implemented practical systems that provide rich functionality over encrypted data: a database system (CryptDB [1]), a web application framework (Mylar [2]), mobile systems (PrivStats [3] and VPriv [4]), and a cloud storage system (CloudProof [5]). My main insight on how to achieve practicality is as follows. Striving to use one generic encryption scheme for all possible computations as in FHE results in prohibitive inefficiency. Instead, one should focus on a class of applications that is broad, yet it has some common traits; for example, SQL databases are used by many applications, yet SQL has certain common operations. Then, from these common traits, one can extract a few primitive operations using which one can support most computations performed by this class. Next, one designs or uses encryption schemes that efficiently support each of these primitives, resulting in supporting most computations in that class efficiently over encrypted data; finally, one builds a system that makes these encryption schemes work together, which often times requires devising new systems techniques.

Adding cryptographic expertise to systems. To implement the insight above for the systems mentioned, I contributed new system designs as well as constructed new encryption schemes: mOPE [6] and ADJ-JOIN [7] for order and join operations in CryptDB, and multi-key search [8] for searching in Mylar. I also constructed functional encryption schemes [9, 10] that enlarge the class of computations in CryptDB to all functions.

More generally, I bring cryptographic expertise to systems and I believe that cryptography can benefit systems significantly. Using only existing cryptography greatly limits the solutions one can provide. For example, CryptDB needed new encryption schemes for order operations and join operations – both order comparison and join being common operations in a database –, and Mylar needed a new kind of keyword search, search over data encrypted with different keys. Without these new schemes, both these systems would have been far less useful (less practical or less secure).

¹See R. L. Rivest, L. Adleman, M. L. Dertouzos, *On data banks and privacy homomorphisms* (1978) and C. Gentry, *A fully homomorphic encryption scheme* (2009).

Main Projects

I will now summarize the systems I built to protect confidentiality against attackers with full access to server data; then, I will mention my theoretical work on functional encryption, which generalizes the computation in some of these systems.

♦ **CryptDB: Database confidentiality.** I designed and built CryptDB [1, 11], a practical database management system (DBMS) that protects the confidentiality of data in databases: the database (DB) is encrypted and the DB server computes SQL queries over the encrypted database without decrypting it. The data and query results are decrypted only at the client application, inside a CryptDB proxy. The threat model [1, 12] is that the DB server is under attack, but the application and the CryptDB proxy are in a trusted location.

Running a DBMS over encrypted data is a challenging task because SQL consists of more than a hundred different operations and because applications expect high performance from DBMSs. To meet such expectations, CryptDB follows the insight I mentioned before: it uses an efficient encryption scheme for each of a set of basic operations (get/put, equality, order comparison, sum, join, and keyword search) and combines the resulting six encryption schemes to support most SQL queries. I designed two of these encryption schemes, mOPE [6] for order comparison and ADJ-JOIN [7] for join, because there did not exist suitable encryption schemes. These six encryption schemes provide a range of tradeoffs between security and functionality: from strong security with little ability for computation to weaker (but well-defined) security with richer functionality.

CryptDB maximizes security for a set of queries issued by the application using a new technique called *onions of encryptions* [1]; at a high level, CryptDB stacks *different* encryption schemes on top of each other while respecting certain functionality and security properties. CryptDB enables data owners to control the level of security on onions and thus protect sensitive fields with strong encryption schemes. When CryptDB cannot support a function (either because it must maintain a certain security level or because the function is too complex), we propose splitting the execution [13] into smaller executions that can be handled by the server and doing some re-encryption and computation at the proxy.

CryptDB is a practical system: it supports all queries from TPC-C, an industry standard benchmark, and decreases throughput by only 26% as compared to vanilla MySQL (a popular SQL DBMS). Crucially, CryptDB requires no changes to existing DBMSs, and almost no change to applications, making it easy to adopt.

CryptDB already has impact. Google recently deployed a system (Encrypted BigQuery) for performing SQL-like queries over an encrypted database following (and giving credit to) CryptDB's design. Their service uses most of the encryption building blocks from CryptDB, as well as rewrites queries and annotates the schema similarly to CryptDB. Researchers at MIT's Lincoln Labs use CryptDB's design for their D4M Accumulo NoSQL engine employing four of CryptDB's building blocks. Also, volunteering users of sql.mit.edu, a SQL server at MIT, are running their Wordpress instances on top of our CryptDB source code. Finally, SAP AG uses CryptDB's design for their SQL-like service.

♦ **mOPE: Order queries on an encrypted database.** I designed and implemented mOPE [6] (*mutable* OPE), a protocol for computing order operations, such as sorting and range queries, on an encrypted database; CryptDB needed such a protocol. To perform order operations efficiently on an encrypted database, the literature has already proposed order-preserving encryption (OPE): an encryption scheme where, if $x \leq y$, then $\text{Enc}(x) \leq \text{Enc}(y)$. Naturally, the ideal security of such a scheme, as defined in the literature, is that the server should learn nothing about the data other than order (which is the functionality desired). Achieving ideal security turned out to be a difficult task: more than a dozen OPE schemes were proposed, but all leaked more than order; even the state-of-the-art scheme leaks half of the plaintext bits. Our scheme, mOPE, is the first to achieve ideal security and is also practical. The insight behind the scheme is actually not from cryptography, but from systems: we observed that, in a database setting, it is acceptable to have a ciphertext depend on a few other ciphertexts in the database and to correct a small number of ciphertexts occasionally, whereas standard encryption schemes do not consider this model. Updating ciphertexts is what we call *mutation*, and we proved it is required to overcome inherent difficulties with standard OPE schemes.

♦ **Mylar: Securing web applications.** CryptDB assumes the application server is trusted, but in some settings, one wants to protect against attacks to the application server as well, such as when hosting a web application server on a cloud. Therefore, I designed and built Mylar, a platform for writing web applications that protects data confidentiality against attackers who compromise all servers: both the application and database servers. Mylar stores encrypted data on the server, and encrypts/decrypts it only in users' browsers using each user's key.

In comparison to CryptDB's setting, two conceptual challenges arise: (1) multi-users/multi-keys: the data that the server must process is no longer encrypted with only one entity's key, and (2) active adversaries: for example, attackers can modify data or insert Javascript in a webpage to extract plaintext data from user browsers.

Mylar enables users to share data securely and dynamically even in the presence of active adversaries at the server, through a new mechanism for distributing and certifying keys. It also ensures that an adversary did not modify client-side application code via a new code certification scheme.

Importantly, Mylar enables common web application functionality. Mylar embraces the recent shift towards client-side applications that permits processing one specific user's data in his own browser. Nevertheless, there are computations that must be run on the server because they access a large amount of data, such as keyword search. Consider that a user has access to n documents, all of which are shared with different users and hence encrypted with different keys. Hence, one needs a searchable encryption scheme that can search over data encrypted with different keys. No such practical encryption scheme existed for our setting, so I designed a **multi-key search** [8] encryption scheme. The scheme allows a server to search an encrypted word over all the documents a user has access to, without the server learning the word searched for or the contents of the documents.

Mylar is practical: porting 6 web applications to Mylar required changing just 35 lines of code on average, and the performance overheads were modest and did not affect user experience.

Mylar is recent work, but it already has a real deployment: securing a medical application of Newton-Wellesley Hospital from Boston. This is a web application collecting private information from patients suffering from the disease endometriosis. We secured the application with Mylar, and the Mylar-enhanced application is now undergoing Institutional Review Board approval and is under alpha deployment.

♦ **PrivStats and VPriv: Securing mobile systems.** I built two systems, PrivStats [3] and VPriv [4], that protect location privacy of users in a mobile system from an untrusted server. The mobile setting consists of sensors or smartphones uploading car movement information or information relevant to a social network to the server. The server can compute useful aggregate statistics in PrivStats (e.g. traffic congestion, popularity of a social location, events) or functions of each client's path in VPriv (e.g., tolling cost in a month) by only handling encrypted data.

♦ **CloudProof: Confidentiality and integrity for cloud storage.** CloudProof [5] is a storage system in which clients can store data at an untrusted cloud provider. CloudProof protects data confidentiality and can detect if a cloud provider violates certain integrity properties on the data as well as construct proofs that such a violation occurred.

General computation using functional encryption.

Functional encryption² is a cryptographic scheme that can be seen as a generalization of the operations in some of the systems above because it allows computing any function over encrypted data. In functional encryption, a data owner (and holder of a secret key) can give a server encryptions $\text{Enc}(x)$ of various inputs x as well as keys sk_f for functions f . Using any sk_f and $\text{Enc}(x)$, an untrusted server can compute $f(x)$, without learning anything else about x .

Functional encryption (FE) provides a more practical model for database computation than its alternative, fully homomorphic encryption (FHE). The reason stems from the fact that the result of the FE computation at the server is $f(x)$ instead of $\text{Enc}(f(x))$ as in FHE. Consider that the server must filter rows using a predicate F and return the matching rows. The client gives the server sk_F and the server applies it to every encrypted row $\text{Enc}(\text{row})$ and learns only $F(\text{row})$: whether the row should be returned or not. With FHE, the server computes $\text{Enc}(F(\text{row}))$ for each row, but cannot tell whether to return the row because the filter result $F(\text{row})$ is encrypted. Since all rows might match, the server should return the whole database, which is impractical. Overall, the insight is that the server *can make decisions based on the data* (e.g., return a row or not) with FE, but not with FHE.

I wanted to understand rigorously what we can hope to achieve in systems: can we support any computation with functional encryption? We devised two functional encryption schemes for general functions, where the model of computation differs. The first scheme is for functions represented as circuits [9]. The second scheme is for functions represented as Turing machines [10], which comes closer to real programs because Turing machines are more realistic than circuits. For example, one can use loops without having to unroll each loop into the maximum number of steps it can run for. In both these schemes, we required that the number of functions to be computed be fixed a priori, a restriction removed by subsequent work. While these FE schemes are not as practical as the systems I built, they provide a proof-of-concept that general functionality is possible in a more practical security model than FHE's model.

We also showed that our first FE scheme is a powerful primitive: using it, we solved a 30-year old question in cryptography, how to **reuse garbled circuits**. Garbled circuits have been used in numerous places in cryptography. In a garbling scheme, the holder of a secret key can garble a function f into a circuit G_f , called the garbled circuit, and can also encode an input x into $\text{Enc}(x)$. G_f hides the function f and $\text{Enc}(x)$ hides x , but using G_f and $\text{Enc}(x)$, an untrusted server can compute $f(x)$, while learning nothing else about f or x . Since Yao's first garbling scheme in 1986, all garbling

²D. Boneh, A. Sahai, and B. Waters, *Functional encryption: definitions and challenges* (2011) formalized the notion of functional encryption.

schemes designed required that the garbled circuit be ran only once (on one input) or else security gets compromised. This was a waste: the effort to garble f into G_f was useful for only one input. Using our scheme, a garbled circuit can be used an arbitrary number of times.

Other research. I also worked on other aspects of security and systems: auditing the results of online voting [14, 15], Byzantine fault-tolerant systems [16], secure network coding [17], and others.

Future Research

I am enthusiastic to continue my work in security and systems, as well as to bridge systems and cryptography. Some areas of future work include:

Practical computation over encrypted data for machine learning and genomics. In many cases, machine learning algorithms run over sensitive data, so running them over encrypted data could protect data confidentiality. A common class of computations for medical data involve matrix manipulations (e.g., approximations to linear regression). CryptDB supports SQL-like operations over encrypted data and is not suitable for such operations. Instead, a preliminary idea is to note that fully homomorphic encryption (FHE) may be a better fit here than in the case of SQL queries, because some matrix manipulations usually have a fixed access pattern that does not depend on the data. However, it turns out that the way a computation is mapped to FHE can result in performance results that differ by orders of magnitude. I already have some preliminary techniques for mapping matrix computations into more efficient FHE evaluations. Overall, I would like to design an easy-to-use language for expressing matrix operations for FHE along with a compiler that can map a computation to an efficient FHE evaluation by performing various optimizations automatically.

Another future project comes from discussions with genomics researchers who want to run learning algorithms on encrypted genomics data in a different setting: the data is owned by different parties that do not trust each other, but would be willing to disclose a global computation result. Generic secure multi-party computation would be too slow here, so I am hoping to discover common computations for genomics that we can run efficiently, similarly to CryptDB's insight. Currently, I am working on a related project: running machine learning classification over encrypted data; our preliminary results show that we can support efficiently three major classifiers used in eight of the most common machine learning algorithms.

Integrity of server results. Most of my work has focused on protecting data confidentiality, but a malicious server can return incorrect results to client queries. This can affect client functionality and even data confidentiality. I am interested in building systems in which clients can check efficiently the correctness of query results. For example, there is no practical database system with such integrity guarantees. There already exist schemes for checking specific computations, but it is not clear how to combine them into a practical system and how to handle multiple users. There are also general schemes that can verify the result of any computation, but these are too inefficient for databases.

Querying “big data” securely. To protect confidentiality of big data, it would be great to encrypt the data and query it as in CryptDB. Due to the large size of the data, big data systems crucially rely on effective compression. Computing on encrypted data is at odds with compression because we can no longer simply encrypt the compressed bulk of data. I would like to build systems that can query encrypted big data efficiently while enabling compression. Some preliminary thoughts are that some deterministic encryption schemes allow certain forms of compression, and that one might be able to create secure versions of existing algorithms that compute on compressed (unencrypted) data.

Statistics over private cloud-based data of many users. An advantage of cloud-based services is that they have access to large amounts of data from various users and can use it to compute useful global statistics such as recommendations or trends. To provide confidentiality, the data of different users can be encrypted with different keys, as in Mylar. However, encrypting the data with different keys poses challenges to computing such statistics, and I would like to develop practical techniques to enable such computation.

Improving client-side security. Much of my work so far removes trust from the server by making the server process encrypted data. The client-side code still has access to the key for decryption and to decrypted data, as well as performs access control operations such as granting other users access to private data. Therefore, I would like to develop techniques for improving client-side security. For example, I would like to ensure that client-side code does not leak data inadvertently by making access control decisions based on unchecked data from the compromised server.

I believe that cloud computing will gravitate towards processing encrypted data whenever possible because of its benefits: both clients and cloud providers are protected against cloud employees or hackers accessing their sensitive

data or the data of their customers, respectively. CryptDB proved that such an approach can be practical and in the few years since its publication, we already started to see such a movement with Google, SAP AG, MIT’s Lincoln Labs, and other companies adopting CryptDB’s approach. Therefore, I am excited to continue building more secure, more practical, and more functional systems that compute on encrypted data.

More generally, I believe that there is a lot of interesting systems security research that can arise from using cryptographic expertise in systems. Most systems work so far limited itself to using basic cryptography (e.g., encryption, signatures, MAC, or hashing), but cryptography is a much more powerful tool, enabling rich functionalities and security guarantees. Moreover, designing new cryptography for a particular systems problem vastly increases the space of possible solutions. Therefore, I would like to bring some of this cryptographic expressivity to practical systems research.

Finally, I remain broadly interested in security, systems, and applied cryptography.

References

- [1] Raluca Ada Popa, Catherine M. S. Redfield, Nikolai Zeldovich, and Hari Balakrishnan. “CryptDB: Protecting confidentiality with encrypted query processing”. In: *ACM Symposium on Operating Systems Principles (SOSP)*. 2011.
- [2] Raluca Ada Popa, Emily Stark, Steven Valdez, Jonas Helfer, Nikolai Zeldovich, M. Frans Kaashoek, and Hari Balakrishnan. “Securing Web Applications by Blindfolding the Server”. In: *USENIX Symposium on Networked Systems Design and Implementation (NSDI) – to appear*. 2014.
- [3] Raluca Ada Popa, Andrew J. Blumberg, Hari Balakrishnan, and Frank H. Li. “Privacy and accountability for location-based aggregate statistics”. In: *ACM Conference on Computer and Communications Security (CCS)*. 2011.
- [4] Raluca Ada Popa, Hari Balakrishnan, and Andrew J. Blumberg. “VPriv: Protecting privacy in location-based vehicular services”. In: *USENIX Security Symposium (USENIX Security)*. 2009.
- [5] Raluca Ada Popa, Jacob R. Lorch, David Molnar, Helen J. Wang, and Li Zhuang. “Enabling security in cloud storage SLAs with CloudProof”. In: *USENIX Annual Technical Conference (USENIX)*. 2011.
- [6] Raluca Ada Popa, Frank H. Li, and Nikolai Zeldovich. “An Ideal-Security Protocol for Order-Preserving Encoding”. In: *IEEE Symposium on Security and Privacy (Oakland)*. 2013.
- [7] Raluca Ada Popa and Nikolai Zeldovich. “Cryptographic Treatment of CryptDB’s Adjustable Join”. In: *MIT-CSAIL-TR-2012-006*. 2012.
- [8] Raluca Ada Popa and Nikolai Zeldovich. “Multi-Key Searchable Encryption”. In: *Cryptology ePrint Archive, 2013/508*.
- [9] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nikolai Zeldovich. “Reusable garbled circuits and succinct functional encryption”. In: *ACM Symposium on Theory of Computing (STOC)*. 2013.
- [10] Shafi Goldwasser, Yael Tauman Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nikolai Zeldovich. “How to Run Turing Machines on Encrypted Data”. In: *International Cryptology Conference (CRYPTO)*. 2013.
- [11] Raluca Ada Popa, Catherine M. S. Redfield, Nikolai Zeldovich, and Hari Balakrishnan. “CryptDB: Processing Queries on an Encrypted Database”. In: *Communications of the ACM (CACM)*. 2012.
- [12] Carlo Curino, Evan Jones, Raluca Ada Popa, Nirmesh Malviya, Eugene Wu, Samuel Madden, Hari Balakrishnan, and Nikolai Zeldovich. “Relational Cloud: A Database Service for the Cloud”. In: *Biennial Conference on Innovative Data Systems Research (CIDR)*. 2011.
- [13] Meelap Shah, Emily Stark, Raluca Ada Popa, and Nikolai Zeldovich. “Language Support for Efficient Computation over Encrypted Data”. In: *Off the Beaten Track Workshop: Underrepresented Problems for Programming Language Researchers*. 2012.
- [14] Javed A. Aslam, Raluca Ada Popa, and Ronald L. Rivest. “On estimating the size and confidence of a statistical audit”. In: *USENIX Workshop on Accurate Electronic Voting Technology (EVT)*. 2007.
- [15] Javed A. Aslam, Raluca Ada Popa, and Ronald L. Rivest. “On auditing elections when precincts have different sizes”. In: *USENIX Workshop on Accurate Electronic voting Technology (EVT)*. 2008.
- [16] James Cowling, Dan R. K. Ports, Barbara Liskov, Raluca Ada Popa, and Abhijeet Gaikwad. “Census: location-aware membership management for large-scale distributed systems”. In: *USENIX Annual Technical Conference (USENIX)*. 2009.
- [17] Raluca Ada Popa, Alessandro Chiesa, Tural Badirkhanli, and Muriel Médard. “Going Beyond Pollution Attacks: Forcing Byzantine Clients to Code Correctly.” In: *CoRR abs/1108.2080*. 2011.