Privacy Enhancing Technologies FS2025
Lecture 1 – Introduction and Fully Homomorphic Encryption

Florian Tramèr

AGENDA

1. Course Pitch
2. Course Logistics
3. What is "Privacy"?
4. Fully Homomorphic Encryption

1 Course Pitch

$$\underbrace{\text{PRIVACY}}_{\text{I}}\ \underbrace{\text{ENHANCING}}_{\text{II}}\ \underbrace{\text{TECHNOLOGIES}}_{\text{III}}$$

I. This is a course about privacy (in case you were wondering...)

II. We'll mostly look at how to *preserve* privacy (although to motivate why this is necessary we'll sometimes look at attacks on privacy as well).

III. This is a technical class, not a policy class. The class will be fairly mathematical.

This class aims to be both your *first* and *last* class on Privacy Enhancing Technologies (PETS). On one hand, privacy is a topic that is only marginally covered in the CS curriculum, and so this might be your first time learning about topics such as Private Information Retrieval (PIR), Zero-Knowledge (ZK) or Differential Privacy (DP). On the other hand, this class will touch on topics that are at the forefront of current research on PETS. After this class, you will have the tools and background to start reading and dabbling in PETS research.

2 What is "Privacy"?

Glad you asked! According to Wikipedia:

> *"Privacy is the ability of an individual or group to seclude themselves, or information about themselves, and thereby express themselves selectively."*

So privacy is about *control*, and *choice*, of how information is used and shared.

In this (computer science) class, we will focus on how we may *compute* on data in a way that preserves privacy. The general setting we'll consider is that there are multiple parties, each with access to some data x_i, and we want to compute some function

$$y_1, \ldots, y_n \leftarrow f(x_1, \ldots, x_n),$$

where each party might get a different output y_i, and we want the computation to reveal "not too much" about the input data.

What it means for a function to reveal "too much" about the data will be central to the formal definitions of privacy we'll see in this class.

This generic setting covers pretty much all forms of modern cryptography and privacy-preserving data analysis. For example:

- **Encryption:** You can think of encryption as a two-party protocol where one party (the sender) inputs a message m and the other party (the receiver) inputs nothing. The receiver gets the message m and the sender gets no output.
- **Zero-knowledge:** One party (the prover) wants to prove to the other party (the verifier) that they know some input x such that $f(x) = 1$. Here the verifier inputs nothing and the prover inputs the witness x; the verifier just learns the output $f(x) = 1$ without learning anything more about x.
- **Private retrieval:** A server has a database D and a client wants to retrieve an index i. The computed function outputs D_i to the client and nothing to the server.
- **Private census:** A government wants to release aggregate statistics about the population by collecting survey data x_i from multiple participants, without leaking information about the individual participants.
- **Private machine learning:** Multiple parties have labeled data $(x_1, y_1), \ldots, (x_n, y_n)$, and the function computes the training of a machine learning model M, which is then shared with all parties.
- And many more…

While the class aims to provide a comprehensive overview of PETS, it is impossible to cover all topics in depth in this class. Here's a non-exhaustive list of topics we *won't* cover:

- Modern two-party computation protocols
- Anonymous and metadata-hiding communication
- Private payments
- Many differential privacy mechanisms
- …

A note on *utility*. Generally when deploying privacy-preserving technology, your competition is a *non-private* primitive that has been optimized to death for speed and utility. That is, in many situations, privacy has been sacrificed over time to make way for performance. Winning privacy back without sacrificing (too much) speed is hard.

So when designing PETS we will put a lot of focus on making the primitives fast and simple enough to deploy. There has been a big shift in the PETS research community in this regard over the past few years. Many of the primitives we'll see (especially on the cryptography side) have been known for decades, but were mainly of theoretical interest. Yet, modern variants of these schemes are now deployed (or close to being deployed) in production. In this class, we will often focus on these modern efficient primitives, over more technical constructions that are of historical interest.

3 Course Logistics

Staff:

- Prof. Florian Tramèr
- Lukas Fluri (Head TA)
- Daniel Paleka
- Jie Zhang
- Michael Aerni
- Kristina Nikolic
- Pura Peetathawatchai
- David Pulido

Website: https://spylab.ai/teaching/pets-f25/

Individual questions: florian.tramer@inf.ethz.ch or lukas.fluri@inf.ethz.ch

Everything else: Use the Moodle forum for questions about course material, homeworks, course policy, etc.

Grading: There will be a midterm (50%) and a final exam (50%).[1] The tentative date for the midterm is November 3 during the lecture time (to be confirmed). The final is on December 15 during the lecture time. **You will be allowed to bring a *single one-sided page* of A4 notes to the midterm, and to the final.**

Lectures and exercise sessions: Lecture notes will be posted online for each lecture (typed up notes, and the transcript of the lecture). *The lectures are also recorded.*

Attendance in lectures or exercise sessions is not mandatory, but is strongly recommended. The exercise sessions will be used to discuss weekly homework problems that cover the material from the lecture. The homeworks might sometimes introduce additional concepts that could be useful in some form for the midterm or final (in which case, they will be re-introduced in the exam).

Prerequisites: This is not an "introductory" course. While we strive to introduce all the necessary background in class, some experience with complexity classes, cryptography, and probabilities will be helpful. We provide a background sheet to refresh these topics.

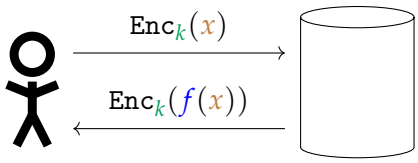
Feedback: This is the second time we teach this class (and the first time with exams), so we would love to get feedback on how to improve it! You can use this form to give us anonymous feedback throughout the semester. You can fill it out as often as you want. And we make mistakes! If something looks wrong or impossible, please let us know.

[1] The course description orginally said 40% and 60%, but we have changed it as both the midterm and final will be 2h long and during the lecture time.

4 Some Fun: Let's Start with Fully Homomorphic Encryption!

Let's start with a simple question: how can we compute on encrypted data? We'll see that this is possible with Fully Homomorphic Encryption (FHE), which is essentially the "holy grail" of cryptography. As we haven't yet introduced a number of technical tools, we'll keep this description somewhat informal. FHE is unfortunately not yet quite practical as it incurs huge computational overhead, but progress is being made!

One motivation for FHE is to securely outsource any computation to a third party:



Input: x Server
Output: $f(x)$

The server learns nothing about the input x and the client gets to learn the output $f(x)$ after decrypting the server's response. For example, you could use this to do private search queries on the Web: you send your encrypted query to Google, and Google uses FHE to compute the result of the query on their end, and then sends back the encrypted result.

Slightly more formally, FHE is an encryption scheme (we'll consider the symmetric case) that consists of four algorithms $(\mathtt{Gen}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{Eval})$ where $(\mathtt{Gen}, \mathtt{Enc}, \mathtt{Dec})$ are defined as in a regular symmetric encryption scheme, and $\mathtt{Eval}$ is an algorithm that takes as input a function f and a vector of ciphertexts $(c_1, \ldots, c_n)$ and outputs a ciphertext $c' = \mathtt{Eval}(f, c_1, \ldots, c_n)$ such that:

$$\mathtt{Dec}_k(\mathtt{Eval}(f, \mathtt{Enc}_k(x_1), \ldots, \mathtt{Enc}_k(x_n))) = f(x_1, \ldots, x_n).$$

As is, this is not very useful as there is a trivial way to achieve this: the server can just add a description of the function f to the ciphertext (so $\mathtt{Eval}(f, c_1, \ldots, c_n) = (f, c_1, \ldots, c_n)$). Then we can define a new decryption algorithm $\mathtt{Dec}'$ that decrypts all ciphertexts and then applies the function f to the results. This is not very interesting. Intuitively, the work should be done by the $\mathtt{Eval}$ algorithm and not by the $\mathtt{Dec}$ algorithm.

Therefore, we require *compactness*: the ciphertext returned by $\mathtt{Eval}$ should not be too large, ideally of size independent of the function f. For now, think of f as a collection of additions and multiplications, and the size of f is just the number of operations. We'll make this more precise in later lectures when we introduce the notion of arithmetic circuits.

Compact FHE is a lot less trivial! In fact, FHE was first proposed as a notion in 1978 by Rivest, Adleman, and Dertouzos [RAD+78], but it was only in 2009 that Gentry [Gen09] introduced the first construction with plausible security (which was then quickly followed by a host of constructions based on standard cryptographic assumptions).

It's worth noting that there are many schemes that support *one* type of homomorphism, e.g., addition *or* multiplication of plaintexts. For example, ElGamal encryption has ciphertexts of the form $\mathtt{Enc}(m) = (g^r, m \cdot h^r)$ where g and h are generators of some group $\mathbb{G}$ and r is a random integer. Then, the (element-wise) product of two ciphertexts is $(g^{r_1} g^{r_2}, m_1 \cdot m_2 \cdot h^{r_1} \cdot h^{r_2}) = (g^{r_1+r_2}, m_1 \cdot m_2 \cdot h^{r_1+r_2}) = \mathtt{Enc}(m_1 \cdot m_2)$. But supporting both additions and

multiplications of plaintexts is a lot more difficult, and implies that we can compute *any* arithmetic function of the plaintexts.

4.1 Somewhat Homomorphic Encryption

We start by constructing a scheme that supports a bounded number of homomorphic operations. Then we see how to extend this to support an unbounded number of operations by using the beautiful idea of bootstrapping.

The scheme below is a pedagogically simple construction due to van Dijk et al. [VDGHV10], for binary circuits (i.e., functions consisting of additions and multiplications modulo 2).

- $\texttt{Gen}()$: Let the key be a "large" odd integer p.
- $\texttt{Enc}_p(b)$: to encrypt a bit $b \in \{0,1\}$, set $c = pq + 2r + b$, where q is a "large" random integer and r is a "small" random integer (much smaller than p).
- $\texttt{Dec}_p(c)$: Output $(c \bmod p) \bmod 2$.

Since we have $2r + b \ll p$, we obtain $c \bmod p = 2r + b$, and so $\texttt{Dec}_p(c) = b$.

Security: The security of the scheme is roughly based on the difficulty of recovering the key p given integers of the form $pq_i + e_i$ where e_i is a small amount of random noise. If there were no noise, it would be easy to recover p by computing the GCD of different ciphertexts. But computing a "noisy GCD" is assumed to be hard if the parameters are large enough.

Homomorphic properties: To add or multiply plaintexts, simply add or multiply the ciphertexts.[2] To see that this decrypts correctly, let $c_1 = \texttt{Enc}_p(b_1)$ and $c_2 = \texttt{Enc}_p(b_2)$. Then:

- $(c_1 + c_2 \bmod p) \bmod 2 = ((b_1 + b_2) + 2(r_1 + r_2) \bmod p) \bmod 2 = b_1 + b_2 \bmod 2$.
- $(c_1 \cdot c_2 \bmod p) \bmod 2 = ((b_1 \cdot b_2) + 2\,(b_1 r_2 + b_2 r_1 + 2 r_1 r_2) \bmod p) \bmod 2 = b_1 \cdot b_2 \bmod 2$.

The homomorphic operations clearly make the noise grow (especially multiplications). So after a bounded number of operations, the noise will overflow p and decryption will fail.

4.2 Bootstrapping

The genius idea of Gentry is to extend a somewhat homomorphic scheme to a fully homomorphic scheme by *homomorphically evaluating the scheme's own decryption algorithm*. What???

Let c be an encryption of a plaintext b with a large amount of noise. Consider the function $g(k) = \texttt{Dec}_k(c)$, which takes as input a key k and uses it to decrypt the ciphertext c. Further let $\texttt{Enc}_p(p)$ be an encryption of the secret key p under itself. Then, we can compute:

$$c' \leftarrow \texttt{Eval}(g, \texttt{Enc}_p(p)),$$

which is also an encryption of b since $\texttt{Dec}_k(c') = g(k) = \texttt{Dec}_p(c) = b$.

But the level of noise in c' is now independent of the level of noise in c! It only depends on the number of operations in the function g and the noise in the encryption of p. So as long as

[2] This is actually not compact as the ciphertext size grows with the number of iterations. We can make it compact by reducing integers modulo kp for some large k after each operation.

the decryption function g is sufficiently simple, we can homomorphically evaluate it within the number of operations that our somewhat homomorphic scheme can support.

Specifically, suppose the somewhat homomorphic scheme can support K homomorphic operations before the noise grows too large. And the decryption function g consists of L operations. Then we can homomorphically evaluate *any* function as long as $K > L$. Basically, we do $K - L$ operations of the function f we want to compute, and then we use bootstrapping to "refresh" the ciphertext so that it can be used for another K operations, and so on.

References

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. Stanford university, 2009.

[RAD+78] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[Rog15] Phillip Rogaway. The moral character of cryptographic work. *Cryptology ePrint Archive*, 2015.

[VDGHV10] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29*, pages 24–43. Springer, 2010.

A Why Care About Privacy Enhancing Technologies?

What would the world look like if we couldn't expect any privacy for some of our communications or interactions. The world (probably) wouldn't end, but it would not be a very nice place either. As beautifully put by Philip Rogaway [Rog15]:

> *"Cryptography rearranges power: it configures who can do what, from what."*

And yet, our privacy is under constant threat.

Many governments are not happy with the idea of encryption they cannot break, and push for legislation to ban (or backdoor) end-to-end encryption.[3] You might say "*but I have nothing to hide!*", and sure, you're not a terrorist or criminal. But the government may one day criminalize your decision to have an abortion[4], or to protest that government...[5]

And encryption is just the tip of the iceberg. Even if your communications are encrypted, your online activity and communication patterns are still tracked wherever you go. Social media apps might know so much about you that they inadvertently (or on purpose) leak your health status[6] your sexual relationships,[7] or the location of your army base![8]

Such metadata has been used by governments and companies to target people when they cannot read their (encrypted) messages, sometimes to fight terrorism[9], but also to target people based on their sexual orientation[10], or (again...) for their health choices.[11]

The rapid progress of AI doesn't help either. The data you post online now powers large-scale facial recognition systems[12] and fuels deepfake crimes ranging from non-consensual pornography[13] to phone or video scams targeting your friends and family.[14]

When companies "try" to protect data, they also often get it wrong. The field of data "anonymization" is littered with examples of data releases that were not. E.g., data of taxi cab rides in New York City which leaked drivers' incomes and home addresses,[15] or the re-identification of Netflix customers from released movie ratings.[16]

Sounds bleak? Well this class will try to show what we* can *do (on the technological side at least) to protect our privacy while still doing cool computery things.

[3] https://www.forbes.com/sites/digital-assets/2024/05/07/european-threat-to-end-to-end-encryption-would-invade-phones/

[4] https://www.washingtonpost.com/technology/2022/07/03/abortion-data-privacy-prosecution/

[5] https://cybernews.com/news/how-encrypted-messaging-changed-the-way-we-protest/

[6] https://splinternews.com/facebook-recommended-that-this-psychiatrists-patients-f-1793861472

[7] https://www.cbsnews.com/sanfrancisco/news/uber-crunches-user-data-to-determine-where-the-most-one-night-stands-come-from/

[8] https://www.wired.com/story/strava-heat-map-military-bases-fitness-trackers-privacy/

[9] https://abcnews.go.com/blogs/headlines/2014/05/ex-nsa-chief-we-kill-people-based-on-metadata

[10] https://www.politico.com/news/2024/02/13/planned-parenthood-location-track-abortion-ads-00141172

[11] https://www.vox.com/recode/22587248/grindr-app-location-data-outed-priest-jeffrey-burrill-pillar-data-harvesting

[12] https://www.nytimes.com/2020/01/18/technology/clearview-privacy-facial-recognition.html

[13] https://www.techtimes.com/articles/301757/20240218/deepfake-dangers-rise-ai-generated-pornography-sparks-global-concerns.htm

[14] https://edition.cnn.com/2024/02/04/asia/deepfake-cfo-scam-hong-kong-intl-hnk/index.html

[15] https://www.theguardian.com/technology/2014/jun/27/new-york-taxi-details-anonymised-data-researchers-warn

[16] https://www.wired.com/2007/12/why-anonymous-data-sometimes-isnt/