Berkeley, Fall 2017, CS261, Week 2
Techniques in Computing on Encrypted Data

Scriber: Weikeng Chen

August 29, 2017

1 Problem description and model

As shown in Figure 1, we want to compute data on untrusted platforms. There are *at least* two approaches: (1) hardware enclaves [MAB+13, BPH15, SCF+15, ZDB+17]; (2) computation over encrypted data with cryptography. Today, we focus on the second part.

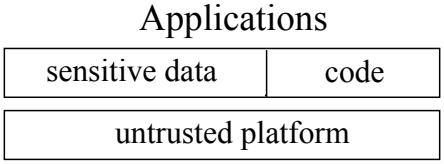


Figure 1: The problem of computing data on untrusted platforms.

1.1 Generic model for outsourced computation

The generic model explains how resource-limited clients outsource the computation. The client encrypts the data and sends to the server, who runs a function f on the data and then returns. Different from plaintext computation, the function f is evaluated on the ciphertexts. There are two parties:

- **Client**: The client is the data owner. It is considered a resource-limited party so that, instead of computing locally, it chooses to outsource the computation [PTK13]. To keep data confidential, the client encrypts the data before sending them to the server. It will receive the encrypted result from the server.
- **Server**: The server has computation capacity to process the (encrypted) data. It is untrusted. It will receive the encrypted data from the client, perform a function f over it, and send back.

An example of the protocol is below. Initially, the client has the key. The key can be a symmetric key SK, or a pair of asymmetric keys SK, PK. No decryption key is revealed to the server.

Client		Server
$\mathtt{data}, \mathsf{PK}, \mathsf{SK}$		f, PK
$\mathbf{Enc}(\mathtt{data})$	$\xrightarrow{\mathbf{Enc}(\mathtt{data})}$	
		$\mathbf{Eval}(f, \mathbf{Enc}(\mathtt{data})) = \mathbf{Enc}(f(\mathtt{data}))$
	$\xleftarrow{\mathbf{Enc}(f(\mathtt{data}))}$	
$\mathbf{Dec}(\mathbf{Enc}(f(\mathtt{data}))) = f(\mathtt{data})$		*(sometimes, we reveal $f(\mathtt{data})$ to the server, like in GC and FE.)*

1.2 Example of addition over ciphertexts using Paillier cryptosystem

Paillier cryptosystem [1] is a public-key encryption with additive homomorphism [Pai99] [2]. It can be used for addition. We denote the two *integer* numbers x_1 and x_2. The addition we want over the plaintexts is the general integer addition. Then, if we encrypt both numbers to $\mathbf{Enc}(x_1)$ and $\mathbf{Enc}(x_2)$, and perform a general integer *multiplication* under the modulus $\mathsf{PK.n}$ on the encrypted data, we have the *addition* result.

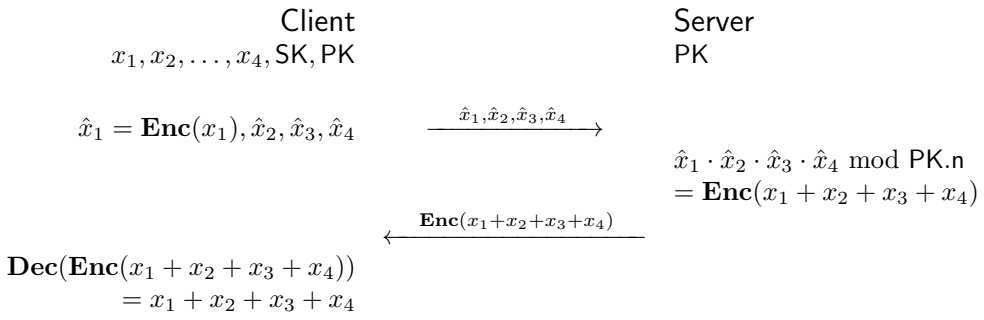
$$\mathbf{Enc}(x_1) \cdot \mathbf{Enc}(x_2) \bmod \mathsf{PK.n} = \mathbf{Enc}(x_1 + x_2)$$

Underlying reason. The underlying reason is that $\mathbf{Enc}(x_1)$ is of the structure $[g^{x_1}]$, and $\mathbf{Enc}(x_2)$ of $[g^{x_2}]$. The multiplication of two ciphertexts resembles the multiplication of two exponentiation values under the same base, and we have $\mathbf{Enc}(x_1 + x_2)$ which is of the structure $[g^{x_1+x_2}]$.

More details. Some students are curious about how Paillier integrates the indeterministic encryption and the additive homomorphism, please check [Pai99] for details. Some homomorphic systems, for example, RSA, cannot achieve both indeterministic encryption and (multiplicative) homomorphism, we clarify.

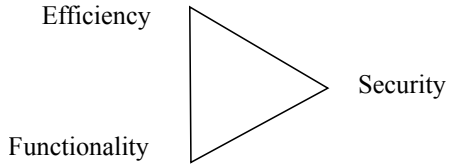
1.3 Secure aggregation via additive homomorphic encryption (AHE)

Below is a protocol to securely aggregate more than two numbers.



2 Tradeoff between security, efficiency, and functionality

The challenge in security is how to balance security, efficiency, and functionality. The ideal functionality we want is (1) except some public knowledge (e.g., size, function), data are not leaked; (2) fast to meet the practical demand; (3) can run generic (arbitrary) function.



Example of $=$ and $\leq$. To support matching and comparison between numbers, we often need to make the tradeoff between efficiency and security. Using the deterministic encryption (DET) and order-preserving encryption (OPE), as in CryptDB [PRZB11], we can process as responsive as if they are unencrypted. However, such property-preserving encryption is vulnerable to statistical analysis, as they are not semantic secure. To increase the security, we need to sacrifice the efficiency.

[1] The pronunciation of Paillier: `https://www.howtopronounce.com/french/paillier/`.

[2] The original paper [Pai99] gives multiple variants, not all are additive homomorphic.

2.1 Common security assumptions

This section presents two commonly used security assumptions. It is crucial for any secure system, including the class project, to declare its threat model using the terminology. We emphasize,

Don't forget about a threat model.

- **Honest-but-curious**. Also known as passive attacker in the system community. This adversary only watches, does no modification, like an eavesdropper. In summary, (1) see everything on the server; (2) no active modification; (3) follow the protocol.
- **Malicious**. Also known as active attacker in the system community. This sort of adversary can do everything. It can return the incorrect result, modify the data. Even if some attacks are detectable, the adversary may still do.

More details on security assumptions can be founded in [AL07, HL10].

3 Fully homomorphic encryption (FHE)

Fully homomorphic encryption is full and homomorphic. (1) "`homomorphic`" means that an operation applied to ciphertexts results in an operation to the plaintext inside the ciphertext; (2) "`full`" means that it supports a generic function. The word "`homomorphism`" comes from group theory in mathematics.

Besides the fully homomorphic encryption, there is also *partial homomorphic encryption*, which means that not all the function can be evaluated under that cryptosystem. For example, Paillier, mentioned in Section 1.2 and 1.3, is additive homomorphic encryption.

History. In 1978, Rivest, Adleman, Dertouzos first explored the concept of fully homomorphic encryption in [RAD78]. In 2009, Gentry proposed the first construction for fully homomorphic encryption [Gen10, Gen09]. This is great because arbitrary functions can be evaluated over encrypted data, with semantic security.

Semantic security. Broadly speaking, semantic security means that ciphertexts leak nothing except the length. If an adversary receives two encrypted messages and the adversary has no secret key, the only thing the adversary can learn is the length.

Efficiency. At the beginning when [Gen09] came, Gentry estimated that a Google search using FHE needs one year. Fortunately, in the recent year, there is a dramatic improvement. But the slowdown can reach $\geq 10^6$ times than plaintext evaluation, depending on the application.

Remark: leakage of length in semantic security. An unsatisfying point in the definition of semantic security is the leakage of length. Even if a cipher perfectly hides the content, it still cannot hide the length. However, the length can be leveraged in many attacks.

- **Example: traffic analysis**. Assume a system with semantic-secure encryption has two remote users, Alice and Bob. They communicate to the server via Internet. Alice's access retrieves much more data than Bob's access (the length of output to Alice is bigger). Then, this pattern can be detected.
- **Approach: padding**. The system can decide a fixed length l and pad every access into this length. l needs to be the maximal possible length. This approach has limitation. If the length varies dramatically, and the worst case length is high (in some settings, e.g., in searchable encryption), then it results in a big performance overhead.

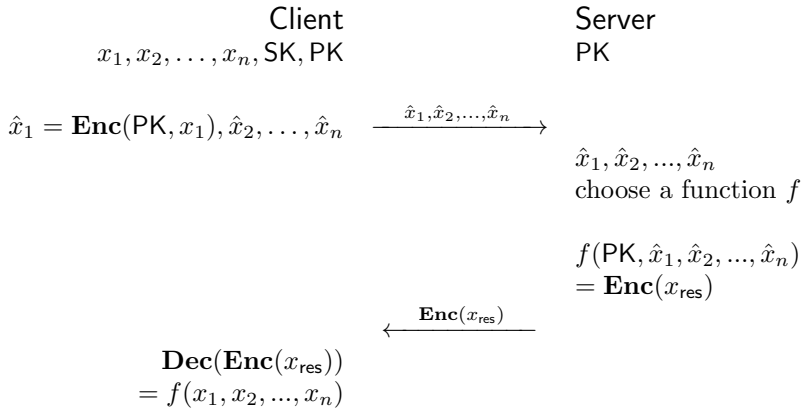
3.1 Syntax, Abstractly

The syntax of FHE is described as $\mathsf{FHE} = \{\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}\}$.

- $\mathsf{KeyGen}(1^\lambda) \to \mathsf{PK}, \mathsf{SK}$ [3]. The key generation algorithm takes as input a unary security parameter 1^λ, and outputs a public key PK and a private key SK.
- $\mathsf{Enc}(\mathsf{PK}, x) \to \hat{x}$. The encryption algorithm outputs the ciphertext $\hat{x}$ for the input plaintext x.
- $\mathsf{Dec}(\mathsf{SK}, \hat{x}) \to x$. The decryption algorithm outputs the plaintext x to the input ciphertext $\hat{x}$.

3.2 Protocol

The protocol below shows how the server computes the function f over the encrypted data.



Example: spam filter. If we want to have an email server to host encrypted emails for many clients, and run a spam filter over encrypted emails with FHE, which only has the public key PK, cannot be realized. If such a spam filter can be built over encrypted data (without an interactive protocol with clients), the spam filter breaks the semantic security of FHE. Even though a generic function can be executed over the encrypted data, the result remains encrypted, and only the party with decryption key knows the result.

3.3 Hide the function f

In the protocol above, the server knows the function f to be executed, to hide the function, we can **make the actual function become part of the encrypted input**. The server is running a public program $\mathcal{F}$, which is the *universal function/circuit interpreter*, similar to a simulator.

Besides the encrypted inputs $\hat{x}_1, \ldots, \hat{x}_n$, the public program $\mathcal{F}$ takes the encrypted function $\hat{f}$ as input. The public program should execute the function over the encrypted data. Details about this construction can be found in [GKP+13].

$$\mathcal{F}(\mathsf{PK}, \boxed{\hat{f}}, \hat{x}_1, \ldots, \hat{x}_n) = f(\mathsf{PK}, \hat{x}_1, \ldots, \hat{x}_n)$$

where $\mathcal{F}$ is the universal function/circuit interpreter.

3.4 Enforce the correct evaluation

In the protocol, the program is executed by the server, what if the server deliberately runs the wrong steps? By nature, FHE is malleable and cannot provide any verifiability. Some related cryptographic treatments for verifiability are authenticated data structures (ADS) [KFPC16, CW11, GTS01, ZKP15, LHKR10, LHKR06] and zero-knowledge proof like zk-SNARK [BCI+13, BSCG+13, BBFR15, BGMS15].

[3] Many encryption schemes have a security parameter 1^λ as input. If a system wants security guarantee for a longer period, it can choose a higher security parameter. The security parameter λ should also increase as the adversary has more computational capacity. Details of the choice of security parameters can be found in [Key17].

Failed attempt: Sign the result. Similar to the authenticity of remote communication, can we ask the program $\hat{f}$ to sign the output? Note that the program must be encrypted to seal the signing key.

$$\hat{f}(\hat{x}) \quad \Rightarrow \quad \begin{array}{l}\hat{f}^*(\hat{x}) \\ \bullet \text{ run function } \hat{f} \\ \bullet \text{ sign the result of the previous step with a hard-coded signing key}^4\end{array}$$

However, using a universal function/circuit interpreter directly in FHE to realize $\hat{f}^*$, is malleable. The server can break the correctness of the execution of $\hat{f}$, but still has the result signed.

$$\begin{array}{l}\hat{f}^*(\hat{x}) \\ \bullet \text{ run function } \hat{f} \\ \bullet \text{ sign the result of the previous step}\end{array} \quad \overset{\textbf{HACKED}}{\Longrightarrow} \quad \begin{array}{l}\hat{f}^*_{\textbf{HACKED}}(\hat{x}) \\ \bullet \text{ run function } \hat{f}' \text{ instead of } \hat{f} \\ \bullet \text{ sign the result of the previous step}\end{array}$$

Although the server tampers the first part, the wrong result will still be signed in the second part. The underlying reason is that directly using FHE to compute cannot glue together different components.

Treatment: Garbled Circuit (GC). Different from FHE which is malleable by nature, GC provides the authenticity by nature [BHR12], even if privacy guarantee is removed [FNO15]. The underlying reason is that GC can glue together all steps. If we integrate GC with FHE, it is possible to provide the non-malleability and verifiability. See [GKP+13] for more details.

3.5 Make FHE more practical

Low-degree polynomial. If the scope of computation is specified [BGV12], for example, evaluating only low-degree polynomials, we have much better efficiency. An example of low-degree polynomial function is $f(x) = x^2 + x + 1$, which has only the degree 2.

Open-source library. The modern cryptographic scheme of FHE [BGV12] has been implemented by Shai Halevi and Victor Shoup. They release a library in GitHub named HElib [HEl17], which is based on C. The list of API is in [HS14].

3.6 Alternatives to FHE with partial homomorphism

There are some partial homomorphic encryption schemes, which have limited functionality, but often faster. They can replace FHE if they are sufficient for computation.

Paillier. [Pai99] is an additive homomorphic cryptosystem based on the composite residuosity assumption. Also, it relies on the hardness of integer factorization.

ElGamal. [EG85] is a multiplicative homomorphic cryptosystem [5] based on Diffie-Hellman assumption. For homomorphic encryption, the cyclic group in ElGamal is usually $\mathbb{Z}_p^*$ where p is a sufficiently large prime.

BGN. [BGN05] leverages the bilinear pairing, which is commonly slower than Paillier and ElGamal [Ben13], but better than FHE. There are two types of ciphertexts: (1) $\mathbb{G}$ ciphertexts; (2) $\mathbb{G}_1$ ciphertexts. Homomorphism provides the above functionalities:

- **Addition in** $\mathbb{G}$. The multiplication of two $\mathbb{G}$ ciphertexts in $\mathbb{G}$ results in the addition of their plaintexts in $\mathbb{Z}_{q_2}$ (q_2 is a prime chosen in key generation). The scalar multiplication can be done with the addition.
- **Addition in** $\mathbb{G}_1$. Similar to the previous case, but the ciphertexts should be dealt in $\mathbb{G}_1$.
- **Multiplication in** $\mathbb{G}$. The bilinear pairing of two $\mathbb{G}$ ciphertexts results in a new ciphertext in $\mathbb{G}_1$, carrying the multiplication of plaintexts.

[4] Since the program is encrypted, message authentication code (MAC) is also sufficient for single-user case.
[5] Generally, the multiplicative homomorphic cryptosystem has to exclude the zero (0) from the message space.

Note that there is no way to transform an element in $\mathbb{G}_1$ back to $\mathbb{G}$ without the decryption key. Therefore, the degree of polynomials to evaluate is at most two. Another drawback of the BGN cryptosystem is that the decryption needs $O(\sqrt{T})$ steps for the message space $\mathcal{M} = \{1, \ldots, T\}$.

4 Multi-party computation with garbled circuit (GC)

This section introduces the secure multi-party computation (SMPC) and one of the general approaches about the garbled circuit (GC). The beginning of multi-party computation with GC is Yao's paper [Yao82] [6]. Since then, there are many research works on how to improve the efficiency to make it practical.

4.1 Problem definition and examples

Imagine there are three parties: P1, P2, P3, as in Figure 2. Each one of them holds its own private data: x_1, x_2, x_3, respectively. They want to compute a public function $F = F(x_1, x_2, x_3)$ jointly without leaking the private data to other parties, while everyone can receive the output of the function [7]. Some common applications of this include counting the average salary, voting, and auction.

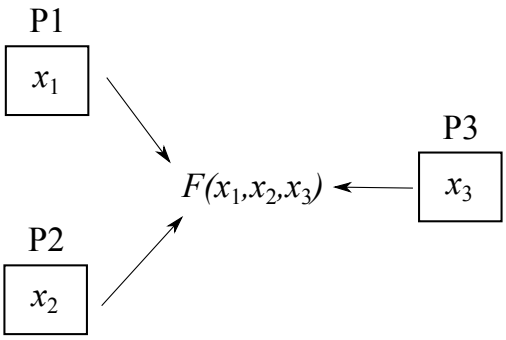


Figure 2: Federated learning from three data providers

Yao's Millionaires' Problem. There are two millionaires, Alice and Bob, have some money. We denote Alice's money as m_{A}, and Bob's money as m_{B}. They want to know who is richer but nothing else [8] This can also be used to compare the salary of whom not a millionaire.

Federated learning. Opaque [ZDB+17] uses hardware enclaves but is still a good example. Everyday, many hospitals have medical data. If they can be securely and legally centralized to make a big database, it can benefit the medical research like cancer. However, the hospitals are not allowed to share the patients' health records to others, in order to protect the privacy. If we can build a secure data warehouse and evaluate the encrypted data, this concern can be resolved. This is the federated learning.

Fraud detection for banks. SecureML [MZ17] uses multi-party computation but only partly with garbled circuit, but it is still a good example. Everyday, there are credit cards being stolen or frauded and people lose money. Different banks can cooperate together to figure out a possible fraud or a malicious merchant. For example, a person with two different credit cards, if they are used in different countries in the same time, there is a high possibility that the card is misused. But banks may not be willing to share the data due to bussiness concerns. Multi-party computation enables these banks to build a model to detect frauds without the sacrifice of data privacy.

[6] Some publications use [Yao86] (worked done at Stanford or Pricenton) instead of [Yao82] (worked done at Berkeley) as the first paper for multi-party computation with garbled circuit, which is not correct.

[7] That means the function F should not be an identity function, otherwise every party learns the input of another party. And the function should not be a constant function, otherwise it is not worth to compute.

[8] It is easy for garbled circuit to output three results $>, =, <$. But not that easy for protocols with homomorphic encryption [BPTG15, Veu11]. Generally for these, we have a binary output $\geq$ or $<$.

4.2 Syntax of GC

A garbling scheme can be described as $\mathsf{GC} = \{\mathsf{Garble}, \mathsf{Encode}, \mathsf{Evaluate}\}$. Details of the construction can be found in [BHR12, ZRE15, BHKR13, KShS12, KMR14, FNO15]. We explain the GC in the setting of outsourced computation in the client-server model, similar to Section 1.3.

- Data owner, client side:
  - $\mathsf{Garble}\left(\mathsf{C}, 1^\lambda\right) \to \mathsf{GC}, \mathsf{SK}$. The algorithm takes as input a boolean circuit C, which is the circuit expression of our desired function $f \in \mathsf{NC}^0$, and a unary security parameter 1^λ. It generates the garbled circuit GC and a secret key for encoding SK. Generally, the garbled circuit hides the functionality to be executed. But in modern practical constructions, the secrecy of C is not crucial.
  - $\mathsf{Encode}\left(\mathsf{SK}, x\right) \to c$. The algorithm takes in the secret key SK and generates the ciphertext c for the input x. The ciphertext is semantic secure that it leaks nothing other than the length, i.e. leak nothing about C and x.
- Evaluator, server side:
  - $\mathsf{Evaluate}\left(\mathsf{GC}, c\right) \to \mathsf{C}(x)$. The algorithm evaluates the garbled circuit GC over the encrypted input c. The server receives the computation result $\mathsf{C}(x)$, where C is the boolean circuit for the function f.

4.3 Security

The Yao's garbled circuit [Yao82] leaks nothing other than the layout of the boolean circuit and the length of input and output. Consider the adversary model of the evaluator (the server, here), the adversary can be either passive or active. Unlike FHE, the program is not malleable, so it works to sign or MAC the output to enforce the correct execution.

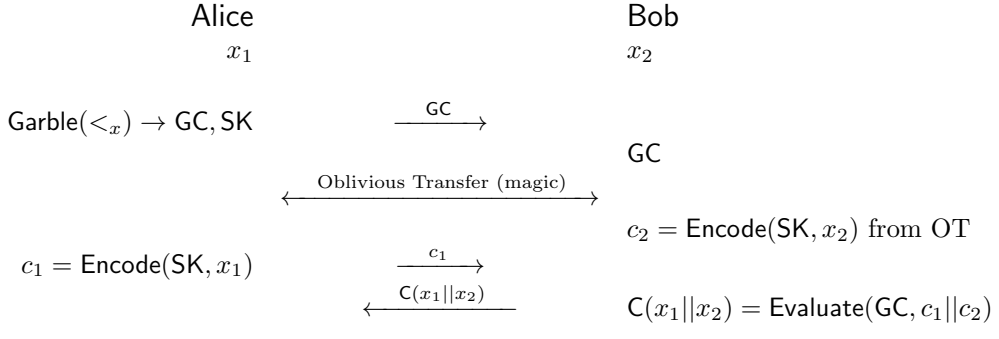
One-time use only. Yao's construction of the garbled circuit, and many practical schemes today, are one-time. It means that the security guarantee *might be* broken if more than one different inputs are generated by the data owner with SK in the algorithm Encode.

4.4 Efficiency

The garbled circuit can be large and slow for complex programs. But it is good for those programs that are easy to be converted into a fixed-depth boolean circuit.

4.5 Protocol for Yao's Millionaires' Problem

Assume that now Alice has money x_1, Bob has money x_2. Both Alice and Bob are modeled as honest-but-curious adversaries. Alice creates a garbled circuit for the comparison function $<_x$, sends this garbled circuit $\mathsf{GC} = \mathsf{Garble}(<_x)$ to Bob. Then Bob makes an oblivious transfer with Alice (explained later) to obtain the encoding of x_2. Alice sends its encoding of x_1. Bob evaluates the circuit and shares the result with Alice.



Brief description of oblivious transfer (OT). Oblivious transfer is a tool to get what you need among a collection of data without revealing which it is. Below are the properties of oblivious transfer. We omit the details about oblivious transfer. Details can be found in [IKNP03, ALSZ13, CO15, CDG+17].

- **Correctness:** Alice has two messages m_0, m_1 (which refers to one bit in the input), Bob wants to obtain m_σ where $\sigma \in \{0, 1\}$.
- **Security:** Bob cannot figure out $m_{1-\sigma}$ and Alice cannot know σ.

4.6 Extension

Reusable garbled circuit. In 2013, the construction of reusable garbled circuit became possible [GKP+13]. It uses different cryptographic tools, including but not limited to, fully homomorphic encryption (FHE), functional encryption (FE), two-outcome attribute-based encryption (ABE^2). The construction is succinct, with a nice complexity, but it is too slow measured by the wall clock, so it is mostly of theoretical interest.

JustGarble that leverages AES-NI. The speed of the evaluation of a garbled circuit depends on the pseudo-random function (PRF). It is known that secure symmetric encryption schemes like AES can be used to build secure PRF [MMO85, BHKR13, ZRE15]. If we can execute a PRF within only a few CPU cycles, we can make the execution of garbled circuit efficient.

Fortunately, many recent Intel CPU processors support AES instructions. The programmer can use these instructions for AES encryption and decryption. Therefore, using the garbling techniques based on AES, we can execute the garbled circuit quickly.

5 Specialized schemes with leakage

To apply cryptography into real-world applications, it must have a sound performance. By allowing a little information leaked to other parties, we can achieve better performance.

5.1 Deterministic encryption (DET)

Consider a database with two columns: `Name`, `Age`. The client wants the functionality to go through all rows that `Age=100`, but also wants to keep the data encrypted.

Name	Age
Enc(Alice)	Enc(100)
Enc(Bob)	Enc(100)
...	...

If the ciphertext for the same plaintext value is the same, the client can submit an unmodified SQL query:

```
SELECT * FROM table WHERE Age = Enc(100)
```

The raw symmetric cipher is a one-one mapping from $\{0, 1\}^\lambda$ to $\{0, 1\}^\lambda$, like AES. So a symmetric cipher is naturally deterministic. To encrypt a varying length of messages, the symmetric cipher uses the mode of operation. **The mode of operation should be selected very carefully.** Because most common modes of operation are leaky. CryptDB [PRZB11] adopts the AES-CMC mode, which leverages double encryption in different directions to remove the prefix consistency. [9] See [HR03] for details.

[9] We can also consider using an indeterministic encryption to store the data, and trivially use ECB mode to encrypt the hash of the message with a collision-resistant hash (CRH).

Efficiency and functionality. The search over deterministic encrypted ciphertexts is as fast as plaintext. However, the functionality is highly limited, because it only supports the equality test.

Security. The deterministic encryption is not semantic secure because two identical plaintexts have the same ciphertexts. This is harmful in some settings but is harmless in some other settings.

- **Insecure**. There are many repetitive plaintexts. The distribution of discrete values helps the recovery of the underlying content. Many data has a distribution that is (accurately or roughly) known by the public. These data include patient disease, age, gender. The gender distribution in CS department is a public knowledge, at least roughly.
- **Secure**. The plaintexts are supposed to be unique. These data can include the Berkeley student ID, social security number (SSN).

5.2 Order-preserving encryption (OPE)

Order-preserving encryption refers to the encryption scheme that:

$$a < b \Longrightarrow \mathsf{Enc}(a) < \mathsf{Enc}(b)$$

This immediately means that OPE is not semantic secure because it leaks the order. Details about the security of OPE can be found in [PLZ13, BPP16]. For most OPE schemes, if the plaintexts are equal, the encryption can still be different. If we compare two ciphertexts with actually the same plaintexts, the result can be $<$ or $>$, due to the randomness of noise.

5.3 Searchable encryption (SE)

Searchable encryption (SE) enables the keyword search over literal texts, such as the digital library and the Internet, without revealing the texts. Consider we have a list of keywords $\{d_1, d_2, ..., d_n\}$, the index is:

$\mathsf{Enc}(d_1)$	$\mathsf{doc}_1, \mathsf{doc}_7, \mathsf{doc}_{15}, \mathsf{doc}_{17}, ...$
$\mathsf{Enc}(d_2)$	$\mathsf{doc}_1, \mathsf{doc}_2, \mathsf{doc}_{12}, ...$
...	...
$\mathsf{Enc}(d_n)$	$\mathsf{doc}_6, \mathsf{doc}_7, \mathsf{doc}_{13}, \mathsf{doc}_{26}, \mathsf{doc}_{32}...$

In the single-keyword case, if the client wants to retrieve the documents associated with the keyword d_i, the client can retrieve the list (on the right side) for the corresponding encrypted keyword $\mathsf{Enc}(d_i)$. There are many constructions with good efficiency and practical for real-world applications. For example, the first paper in searchable encryption is [SWP00] by Dawn Song, David Wagner, Adrian Perrig, all at Berkeley.

The tradeoff between security and efficiency. Searchable encryption is efficient but leaks much side information. For example, the number of documents of a keyword reflects whether or not the word is frequent. There are a line of works [CGPR15, WGL+17, Nav15, ZKP16, IKK12] on leakage-abuse attacks on SE. They do not need to break the security guarantee but leverage the leakage of the search functionality that the scheme provides.

5.4 Framework to prove the security of a leaky system

Broadly speaking, many practical systems have leakage, and sometimes a system without leakage is theoretically impossible. When people prove the security of a leaky system, the following framework is used:

- Analyze the system to see what is leaked.
- Declare the leak function of every leaky action, e.g. $\mathsf{Leak}(DB, (\texttt{RANGE}, [a, b])) = (\texttt{RANGE}, \mathrm{rank}(a-1), \mathrm{rank}(b))$ describes the leakage of range query over the value $[a, b]$ in [BPP16].
- Prove that a probabilistic polynomial-time adversary computationally cannot distinguish the system and an ideal secure system, which achieves the functionality without reasons, obviously without leakage, but it deliberately leaks $\mathsf{Leak}(\cdot)$.

6 Compare with hardware enclaves

This section provides a concise list of properties to compare between the computation over encrypted data and hardware enclaves.

Computation over encrypted data	Hardware enclave
• no practical scheme for realistic generic program • expensive to add integrity protection	• the current deployment is not pervasive [10] • cheap to add integrity with enclaves • must trust the vendor (e.g. Intel) [11] • trust that no hardware attack to CPU is possible • assume no side channel (not fundamental) • assume no exploit to the program inside the enclave • execution is in the processor speed • can run a client in the enclave [12] • constrained in the number of cores • constrained in the memory size (128MB today)

Table 1: Comparison of computation over encrypted data and hardware enclaves

References

[AL07] Yonatan Aumann and Yehuda Lindell. *Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries*, pages 137–156. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS '13, pages 535–548, New York, NY, USA, 2013. ACM.

[BBFR15] M. Backes, M. Barbosa, D. Fiore, and R. M. Reischuk. Adsnark: Nearly practical and privacy-preserving proofs on authenticated data. In *2015 IEEE Symposium on Security and Privacy*, pages 271–286, May 2015.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Omer Paneth, and Rafail Ostrovsky. Succinct non-interactive arguments via linear interactive proofs. In *Theory of Cryptography: 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, pages 315–333, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Ben13] Benchmark of the pairing-based cryptography library, 2013. `https://crypto.stanford.edu/pbc/times.html`.

[BGMS15] Dan Boneh, Divya Gupta, Ilya Mironov, and Amit Sahai. Hosting services on an untrusted cloud. In *Advances in Cryptology - EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 404–436, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In Joe Kilian, editor, *Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005. Proceedings*, pages 325–341, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325, New York, NY, USA, 2012. ACM.

[10] For example, so far, Amazon cloud services do not support SGX. This should be a temporary issue. It is similar to the fact that TPM is not everywhere.

[11] It includes no backdoor and the correct implementation of SGX (the microcode in CPU). However, in real life, we have already trusted the CPU vendors on the client side.

[12] It is promising to run part of the client in the server-side hardware enclave to reduce the network roundtrips and reduce the bandwidth.

[BHKR13] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy*, pages 478–492, May 2013.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 784–796, New York, NY, USA, 2012. ACM.

[BPH15] Andrew Baumann, Marcus Peinado, and Galen Hunt. Shielding applications from an untrusted cloud with haven. *ACM Trans. Comput. Syst.*, 33(3):8:1–8:26, August 2015.

[BPP16] Tobias Boelter, Rishabh Poddar, and Raluca Ada Popa. A secure one-roundtrip index for range queries. Cryptology ePrint Archive, Report 2016/568, 2016. `http://eprint.iacr.org/2016/568`.

[BPTG15] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *The Network and Distributed System Security Symposium 2015 (NDSS 15)*. Internet Society, 2015.

[BSCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *Advances in Cryptology – CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 90–108, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[CDG+17] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. *Laconic Oblivious Transfer and Its Applications*, pages 33–65. Springer International Publishing, Cham, 2017.

[CGPR15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption, 2015.

[CO15] Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. In *Proceedings of the 4th International Conference on Progress in Cryptology – LATINCRYPT 2015 - Volume 9230*, pages 40–58, New York, NY, USA, 2015. Springer-Verlag New York, Inc.

[CW11] Scott A. Crosby and Dan S. Wallach. Authenticated dictionaries: Real-world costs and trade-offs. *ACM Trans. Inf. Syst. Secur.*, 14(2):17:1–17:30, September 2011.

[EG85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[FNO15] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In *Advances in Cryptology - EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 191–219, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

[Gen10] Craig Gentry. Computing arbitrary functions of encrypted data. *Commun. ACM*, 53(3):97–105, March 2010.

[GKP+13] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 555–564, New York, NY, USA, 2013. ACM.

[GTS01] M. T. Goodrich, R. Tamassia, and A. Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. In *DARPA Information Survivability Conference amp; Exposition II, 2001. DISCEX '01. Proceedings*, volume 2, pages 68–82 vol.2, 2001.

[HEl17] Helib, 2017. `https://github.com/shaih/HElib`.

[HL10] Carmit Hazay and Yehuda Lindell. A note on the relation between the definitions of security for semi-honest and malicious adversaries. Cryptology ePrint Archive, Report 2010/551, 2010. `http://eprint.iacr.org/2010/551`.

[HR03] Shai Halevi and Phillip Rogaway. *A Tweakable Enciphering Mode*, pages 482–499. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[HS14] Shai Halevi and Victor Shoup. Algorithms in helib. Cryptology ePrint Archive, Report 2014/106, 2014. `http://eprint.iacr.org/2014/106`.

[IKK12] Mohammad Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *The Network and Distributed System Security Symposium 2012 (NDSS 12)*. Internet Society, 2012.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology - CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings*, pages 145–161. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[Key17] Keylength - cryptographic key length recommendation, 2017. `https://www.keylength.com/`.

[KFPC16] N. Karapanos, A. Filios, R. A. Popa, and S. Capkun. Verena: End-to-end integrity protection for web applications. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 895–913, May 2016.

[KMR14] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. Flexor: Flexible garbling for xor gates that beats free-xor. In *Advances in Cryptology – CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, pages 440–457. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[KShS12] Benjamin Kreuter, Abhi Shelat, and Chih hao Shen. Billion-gate secure computation with malicious adversaries. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 285–300, Bellevue, WA, 2012. USENIX.

[LHKR06] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. Dynamic authenticated index structures for outsourced databases. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 121–132, New York, NY, USA, 2006. ACM.

[LHKR10] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. Authenticated index structures for aggregation queries. *ACM Trans. Inf. Syst. Secur.*, 13(4):32:1–32:35, December 2010.

[MAB+13] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative instructions and software model for isolated execution. In *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13, pages 10:1–10:1, New York, NY, USA, 2013. ACM.

[MMO85] S. M. Matyas, C. H. Meyer, and J. Oseas. Generating strong one-way functions with cryptographic algorithm. *IBM Technical Disclosure Bulletin*, 27:5658–5959, 1985.

[MZ17] P. Mohassel and Y. Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38, May 2017.

[Nav15] Muhammad Naveed. The fallacy of composition of oblivious ram and searchable encryption. Cryptology ePrint Archive, Report 2015/668, 2015. `http://eprint.iacr.org/2015/668`.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings*, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[PLZ13] Raluca Ada Popa, Frank H. Li, and Nickolai Zeldovich. An ideal-security protocol for order-preserving encoding. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 463–477, Washington, DC, USA, 2013. IEEE Computer Society.

[PRZB11] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: Protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 85–100, New York, NY, USA, 2011. ACM.

[PTK13] A. Peter, E. Tews, and S. Katzenbeisser. Efficiently outsourcing multiparty computation under multiple keys. *IEEE Transactions on Information Forensics and Security*, 8(12):2046–2058, Dec 2013.

[RAD78] R L Rivest, L Adleman, and M L Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[SCF+15] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. Vc3: Trustworthy data analytics in the cloud using sgx. In *2015 IEEE Symposium on Security and Privacy*, pages 38–54, May 2015.

[SWP00] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, SP '00, pages 44–, Washington, DC, USA, 2000. IEEE Computer Society.

[Veu11] Comparing encrypted data, 2011. `http://visionlab.tudelft.nl/sites/default/files/Comparing%20encrypted%20data.pdf`.

[WGL+17] L. Wang, P. Grubbs, J. Lu, V. Bindschaedler, D. Cash, and T. Ristenpart. Side-channel attacks on shared search indexes, May 2017.

[Yao82] A. C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 160–164, Nov 1982.

[Yao86] A. C. C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167, Oct 1986.

[ZDB+17] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 283–298, Boston, MA, 2017. USENIX Association.

[ZKP15] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. Integridb: Verifiable sql for outsourced databases. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1480–1491, New York, NY, USA, 2015. ACM.

[ZKP16] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 707–720, Austin, TX, 2016. USENIX Association.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole. In *Advances in Cryptology - EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 220–250. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.