CLOUD ENCRYPTION

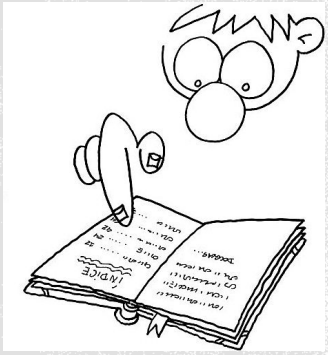
Applied Cryptography

Silvio Ranise [silvio.ranise@unitn.it or *ranise@fbk.eu*]

SECURITY & TRUST

FONDAZIONE BRUNO KESSLER

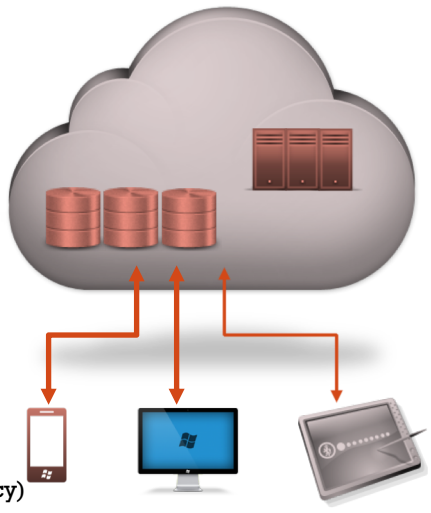
CONTENTS



2 OVERVIEW

CLOUD COMPUTING (1)

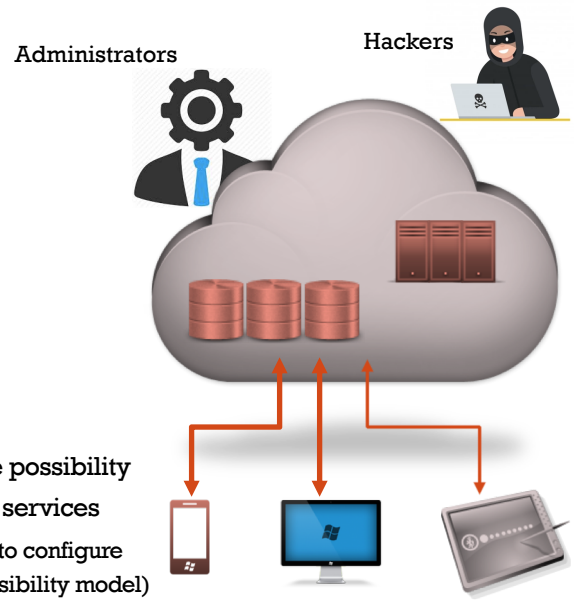
- **Definition**
  - Using remote, Internet-based servers for computational or storage resources
- **Pros**
  - Accessible from anywhere, any time
  - Maintenance done by someone else (unless self-hosted!)
  - Cost-effective, as part of a shared data-center
  - Scalable resources
- **Cons**
  - Availability relies on Internet connection and server reliability
  - Long-distance communication creates a performance hit
  - Your data is on **someone else's machines**
  - Co-located with others - maybe even competitors (multi tenancy)



CLOUD COMPUTING (2)

- **Idea**
  - computational resources that run your programs are located and managed remotely
- **Instances**
  - Amazon EC2 (Elastic Compute Cloud)
    - https://aws.amazon.com/ec2
  - Microsoft Azure
    - https://azure.microsoft.com
  - Google Cloud Platform - Compute Products
    - https://cloud.google.com/products/compute/

CLOUD COMPUTING: SECURITY CONCERNS



- Data in the cloud may be vulnerable to
  - Snooping administrators
  - Hackers with illegal access
    - Compromised servers
- One of the advantages of using the cloud is the possibility to use well-engineered and managed security services
  - It is typically still the responsibility of customers to configure such services appropriately (split/shared responsibility model)

CLOUD COMPUTING AND TRUST (1)

- Services are provided by third parties and not in house…
- … users have no real guarantee that the cloud service providers are doing what they claim to be doing
- **Trust becomes a key issue**
- Example problem: **proof of data storage**
  - If a cloud infrastructure is used for long term storage, then clients may want to know whether their uploaded data is still there
  - Rogue providers could still charge for storage which they have deleted on the hope that the client never asks for the data back
  - The data may be so large that clients simply asking for it to be downloaded every so often for checking may not be feasible
  - Proofs of Storage cryptographic protocols enable a storage provider to prove to client that files are still being stored, without clients needing to keep copies of all that has been stored
    - One such protocol is described in this paper:

https://www.arijuels.com/wp-content/uploads/2013/09/JK07.pdf

CLOUD COMPUTING AND TRUST (2)

- The main security problem with cloud computing is that data loaded into the cloud is no longer in the control of the data creator/data controller and **cloud service providers may be honest but curious**
- The main reason for moving application to the cloud is to enable computation using a fast and cheap cloud infrastructure:
  - there is a conflict between the need to secure and the need to process data
- When the data has to be retrieved, processed or searched, **use of standard encryption is not useful**: one has
  - either to allow the cloud to decrypt the data
    - (this needs trust in cloud service providers)
  - or to download the entire data set to the user's machine
    - (this spoils the benefits of moving to the cloud in the first place)

CLOUD COMPUTING AND TRUST (3)

- Even if processing is not required, one needs some additional cryptographic functionality to be able to search and retrieve data which has been stored (encrypted) in the cloud
- There are a number of emerging technologies which aim to solve this clash of requirements including
  - Fully Homomorphic Encryption (FHE)
  - Multi-Party Computation (MPC)
  - Searchable Encryption
  - Order Preserving Encryption
  - Attribute Based Encryption
  - Delegated Computation
- We give a high-level description of each one and then consider some in more details in the following..

FULLY HOMOMORPHIC ENCRYPTION (FHE)

- Considered one of the recent major break-trough in cryptography
- Introduced by Gentry in his PhD thesis (2009): https://crypto.stanford.edu/craig/craig-thesis.pdf

- **Idea**
  - An arithmetic circuit can be applied to a ciphertext and the result is the encryption of the output of the arithmetic circuit as if it had been evaluated on the underlying plaintext
- **In the cloud**
  - Data owners encrypt their data *D*, to obtain *enc(D)* and send it to the cloud provider
  - Then suppose the data owners wanted to perform some operation *f* on the data *D* to obtain *f(D)*
    - For example *D* could be a data base and *f* could be a search for all items corresponding to a given person
  - The cloud can compute *enc(f(D))* which is then returned to the data owners
  - The data owners then decrypt this ciphertext to obtain *f(D)*
- **Main problem**
  - Efficiency and scalability
  - Only relatively simple circuits can be evaluated efficiently

MULTI-PARTY COMPUTATION (MPC)

- Idea around since 1980s
- An overview can be found at https://www.cs.virginia.edu/~evans/pragmaticmpc/pragmaticmpc.pdf

- **Idea**
  - It allows the equivalent of FHE with the use of multiple servers as opposed to a single one
  - Thus data can be securely outsourced to multiple cloud providers who then can compute on this data, such that one can tolerate a set of colluding adversaries (up to some bound on the number)
- **In the cloud**
  - Data owners split their data *D* into chunks *D1, … , Dn* via a secret sharing scheme
  - The shares are then distributed to *n* distinct cloud providers
  - The cloud providers are now able to compute any function *f(D)*, but they obtain partial results (called shares) that are then returned to the data owners for combining into the final solution *f(D)*
    - [computing shares may require some exchange of information among providers]
  - As long as the data owners trust $n - t$ out of the *n* providers, their data is still kept confidentially where ***t* is the maximum number of adversaries which can be tolerated**

SEARCHABLE ENCRYPTION

- Introduced in 2000
- An overview can be found at https://research.utwente.nl/files/6420885/a18-bosch.pdf

- **Idea**
  - Solve the following problem: a user outsources a database to a server and then wants to query the server to obtain data
  - Question: what does security mean in this context?
  - Answer 1: the database contents should remain private to the client
  - Answer 2: in some cases also the access patterns should be private
- Two variants: public or symmetric key
- In both variants, the encrypted database is augmented with a set of tokens
  - Each token is associated with a keyword
- The database is encrypted with any encryption algorithm, the searchable encryption scheme is solely used to construct and search for the tokens
- Public key variant is vulnerable to selected keywords attack and thus is less adopted
- Symmetric key variant is combined with Oblivious RAM techniques to guarantee also the privacy of access patterns

SEARCHABLE ENCRYPTION & ACCESS PATTERNS

Oblivious RAMs hide the access pattern by making any equal-length sequence of clients' data requests to the server equivalent from the point of view of the curious server

- A server, which maintains a data storage system, can gain information about its users' habits and interests, and violate their privacy, even without being able to decrypt the data that they store
- The server can monitor the queries made by the clients and perform different traffic analysis tasks
- It can learn the usual pattern of accessing the encrypted data, and try to relate it to other information it might have about the clients
  - For example, if a sequence of queries q1, q2, q3 is always followed by a stock-exchange action, a curious server can learn about the content of these queries, even though they are encrypted, and predict the user action when the same (or similar) sequence of queries appears again
  - Moreover, it is possible to analyze the importance of different areas in the database, e.g., by counting the frequency of the client accessing the same data items
  - If the server is an adversary with significant but limited power, it can concentrate its resources in trying to decrypt only data items which are often accessed by the target-user

ORDER PRESERVING ENCRYPTION

- Introduced in 2004
- A systematic approach can be found in https://eprint.iacr.org/2012/624.pdf

- **Idea**
  - It solves a variant of the searchable encryption problem in another way
  - It enables ciphertexts to be ordered in a way which respects the ordering of the underlying plaintexts
  - Thus binary search can be conducted on the ciphertexts
- It had been originally introduced in a naive way in the database community

  Order Preserving Encryption for Numeric Data

  http://rsrikant.com/papers/sigmod04.pdf
  - The approach described here does not meet the usual cryptographic standard in the cryptographic literature
- The key problem is that the security properties are very weak

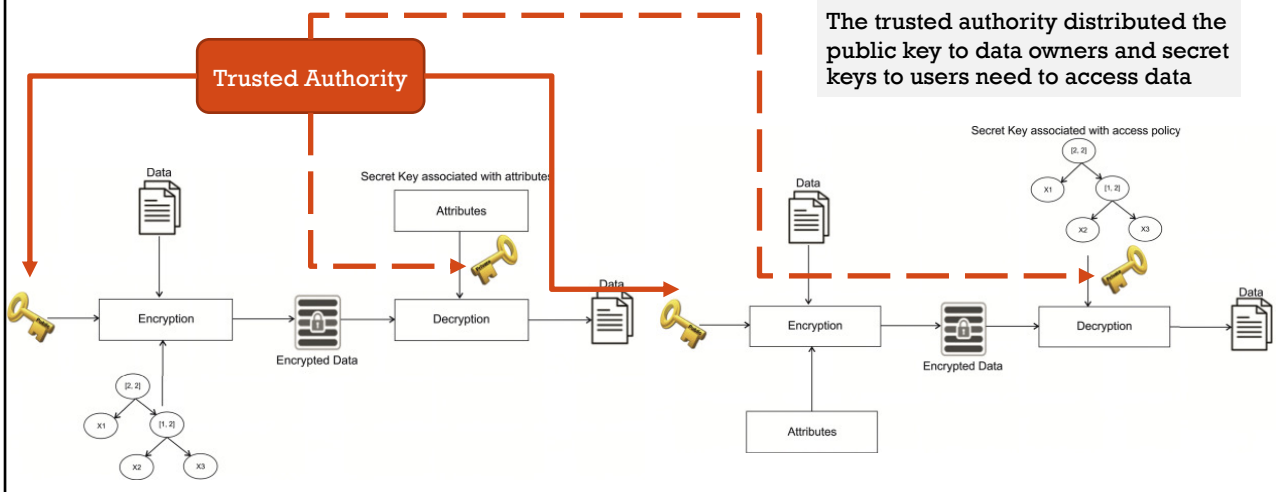
ATTRIBUTE BASED ENCRYPTION

- Introduced in 2005
- A survey can be found at http://ijns.jalaxy.com.tw/contents/ijns-v15-n4/ijns-2013-v15-n4-p231-240.pdf

- **Idea**
  - It extends Identity Based Encryption where decryption of data is allowed on the basis of whether the decryptor satisfies a set of policies associated with given secret keys, which in turn are associated to attributes
- **In the cloud**
  - Complex access control policies for encrypted outsourced data can be embedded within the ciphertext itself
  - This means that a data owner does not need to trust the cloud provider to implement a policy they require

- Identity-based encryption is a type of public-key encryption in which a user can generate a public key from a known unique identifier such as an email address and a trusted third-party server calculates the corresponding private key from the public key
  - In this way, there is no need to distribute public keys ahead of exchanging encrypted data
- The sender can simply use the unique identifier of the receiver to generate a public key and encrypt the data
- The receiver can generate the corresponding private key with the help of the trusted third-party server

ATTRIBUTE BASED ENCRYPTION IN PICTURES



Ciphertext Policy Attribute Based Encryption
The ciphertext is based on access policies and user's secret keys based on attributes

Key Policy Attribute Based Encryption
The ciphertext is based on attributes and the user's secret keys are based on access policies

DELEGATED COMPUTATION

- Introduced in 2005
- A survey can be found at http://ijns.jalaxy.com.tw/contents/ijns-v15-n4/ijns-2013-v15-n4-p231-240.pdf

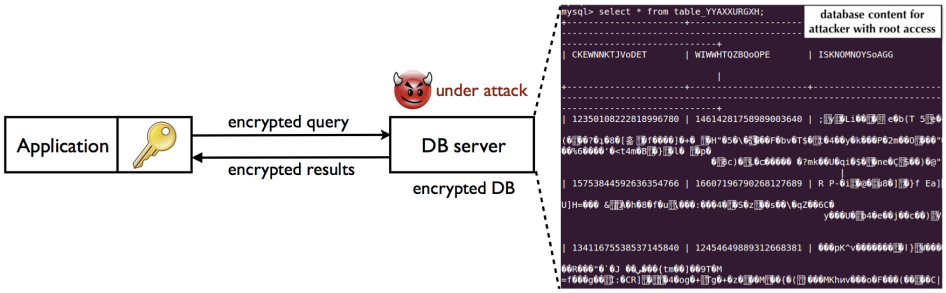
- **Idea**
  - A resource constrained client can delegate a computation to a powerful service provider
  - The client can check whether the provider has performed the correct computation
  - Two running modes
    - non-private: cloud server learns the clients data
    - private: the input data, and perhaps the function, are kept private
- Theoretical results that combine techniques from FHE, ABE, and MPC

MATURITY LEVELS

Technology	Ready for Deployment	Short Term Research Needed	Longer Term Research Needed
Fully Homomorphic Encryption	×	×	√
Multi-Party Computation	√	√	×
Searchable Encryption	√	√	×
Order Preserving Encryption	×	×	√
Attribute Based Encryption	×	√	×
Delegated Computation	×	×	√

AVAILABLE SYSTEM: CRYPTDB

http://css.csail.mit.edu/cryptdb/



- It uses standard cryptography, Symmetric Searchable Security, and Order Preserving Encryption
- The system is efficient but suffers from inherent leakage of information
- It provides a layer of weak security to an outsourced database application, as opposed to providing truly strong security

19 CLOUD STORAGE

CLOUD STORAGE (1)

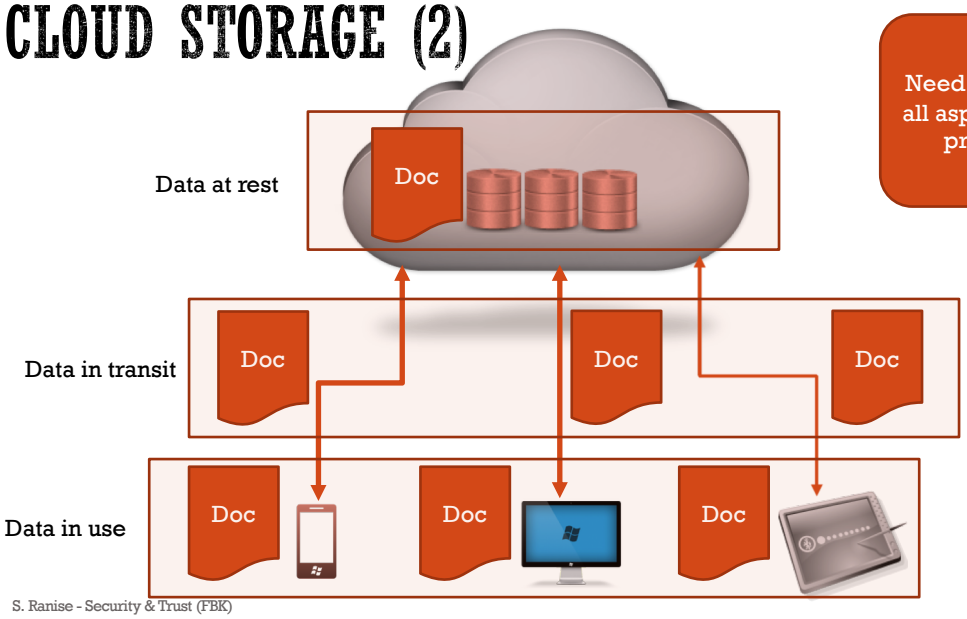
- **Idea**
  - your data/files are stored on a remote server
  - (not a computational service, just storage)
- **Two main models**
  - **File sync**
    - All files on local sys and server
      - Both monitored for changes
      - Local file change → Upload
      - Server file change → Download
  - **File stream**
    - Files not generally on local system
      - File-by-file access
      - Limited local storage during use
      - Download/use/upload

Dropbox

Desktop sync client
- Released June 2007
- Broad support
- Personal, premium, and
- business accounts
- Over 500 million users

Google drive
- Integrated with GSuite
- Broad support
- Personal and
- business accounts
- Supports huge files (5TB)
- Over 800 million users

CLOUD STORAGE (2)



Need to consider all aspects of data protection

CLOUD STORAGE FOR ENTERPRISES

- Conflicting requirements
  - need to keep data confidential with respect to the curious but honest cloud service provider
  - support controlled sharing of data, i.e. employees should be able to selectively access data stored in files to be able to perform their jobs
- Using standard encryption in this context is not obvious since enterprise wants to define the access control policies to files containing its data
- Another difficulty is that access control policy enforcement needs to be distributed
  - Better to avoid central access control module (also to avoid scalability issues)
- **Idea**
  - Use cryptography to enforce access control policies so that sharing among employees is permitted while protecting against curious service providers

ROLE BASED ACCESS CONTROL IN A NUTSHELL

- Role based access control uses role hierarchies to make the specification of policies more compact
- These can always be compiled away without loss of generality

UA: user-role assignment

User	Role
u1	r1
u2	r1
u2	r3
u3	r2

Role	Permission
r1	p1
r2	p2
r2	p4
r3	p3

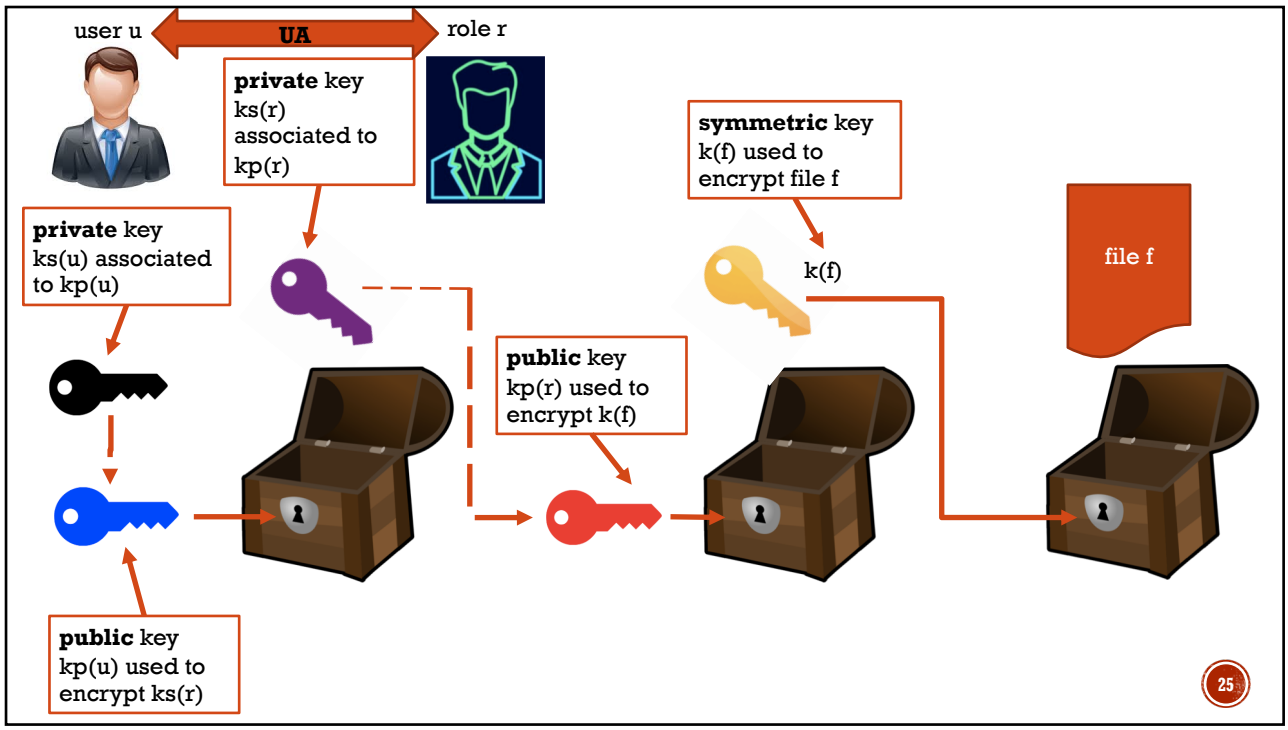
PA: permission-role assignment

User u has permission p iff there exists role r such that (u,r) in UA and (r,p) in PA

A permission is a pair (*a*,*f*) where
- *a* is either *read* or *write*
- *f* is a file identifier

HYBRID CRYPTOGRAPHY (1)

- Each user *u* is equipped with a pair of private and public key: *(ks(u),kp(u))*
- Each role *r* is equipped with a pair of private and public key: *(ks(r),kp(r))*
- Each file *f* is encrypted with a unique symmetric key: *k(f)*
- If *(u,r)* in UA, then compute *{ks(r)}kp(u)*
  - i.e. encrypt the secret key *ks(r)* associated to *r* with the public key *kp(u)* associated to *u*
  - In this way, *u* will be able to retrieve the secret key associated to the role by using its private key
- If *(r, (read,f))* in PA, then compute *{k(f)}kp(r)*
  - i.e. encrypt the symmetric key *k(f)* associated to the file *f* with the public key of the role *r*
  - In this way, a user *u* with read permission on f (i.e. such that (u,r) in UA and (r,(read,f)) in PA) will be able to retrieve the symmetric key associated to *f* by first retrieving the secret key *ks(r)* associated to the role which can then be used to retrieve the symmetric key of the file *f* which is encrypted with the public key *kp(r)* associated to *r*



HYBRID CRYPTOGRAPHY (2)

- Notice that further auxiliary data (e.g., files version numbers and digital signatures), together referred to as metadata, are needed to enforce policies
- Users store their private keys in secure personal devices (e.g., laptops provided with an antivirus) access to which is protected through passwords or similar authentication techniques
- Both encrypted data and metadata are stored in the cloud or in a secure area within the organization
- To write on a file *f*, a user performs the same operations to obtain the symmetric key *k(f)* which is used to encrypt the new file
- An entity (usually called Reference Monitor) checks whether the user has actually write permission before accepting the new file and storing it in the cloud

27

AN IMPORTANT CLASS OF CLOUD STORAGE: CLOUD DATABASES

Also known as **Data Base-as-a-service** (**DBaas**)

DBAAS

- Example: course data-base

Student

StudentId	Name	Addr	GPA	CreditCard	…

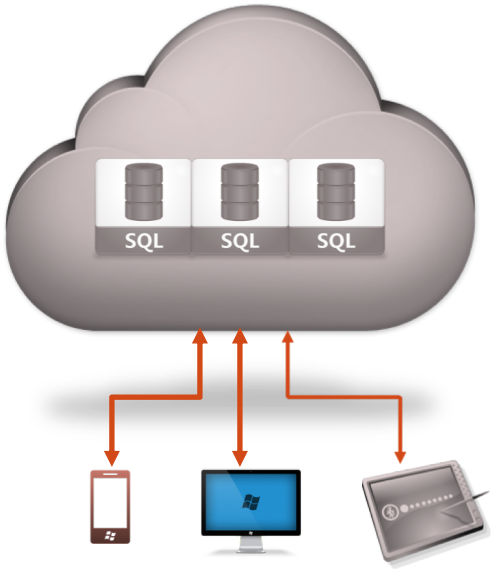
Course

CourseId	Name	InstrId	…

StudentCourse

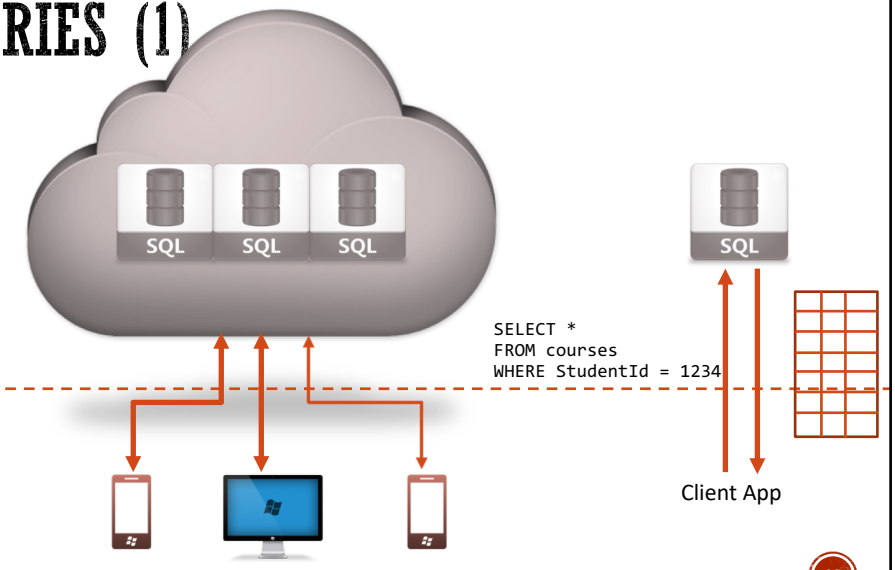
CourseId	StudentId	Grade	…

Sensitive attributes are circled in red



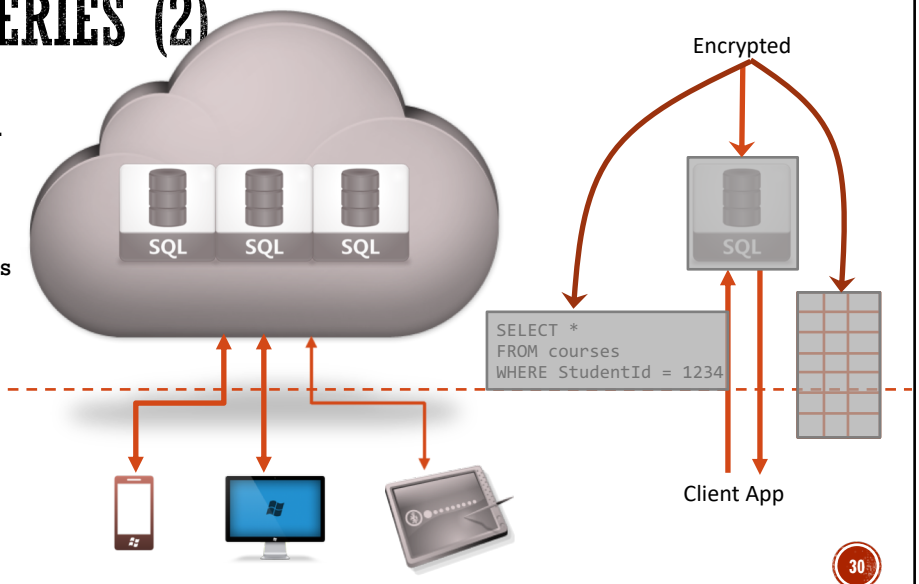
DBAAS QUERIES (1)

- **Plaintext**
- Neither security nor privacy



DBASS QUERIES (2)

- **Using encryption**
- Desiderata
  - encrypted data stored
  - encrypted queries
  - encrypted results



DBAAS SECURITY & PRIVACY

- How we can build DBaas that support encryption in the sense above?

- **Focus**
  - End-to-end systems
  - Trade off on security, performance, and generality

- **Goal**
  - how to use cryptography as a black-box (mostly) and build secure and privacy-preserving DBaas
  - Mainly consider homomorphic encryption schemes

ADVERSARIAL MODEL

- **Passive** adversary
- **Honest but curious** cloud service provider
- Both types of adversaries alter
  - neither content of the database
  - nor the results of queries
- We will not consider active adversaries
  - in other words, only confidentiality is relevant for our discussion, integrity is disregarded

Consider the following query

```
Select Sum (Score)
From Assignment
Where StudentId = 1
```

Result is 139=68+71

FROM

StudentId	AssignId	Score
1	1	68
1	2	71
3	4	99
...	...	...

How can we obtain the same result on the encrypted database? Notice that the result should also be encrypted

TO

Assignment

```
a7be1a6997ad739bd8c9ca451f618b61
b6ff744ed2c2c9bf6c590cbf0469bf41
47f7f7bc95353e03f96c32bcfd8058df
```

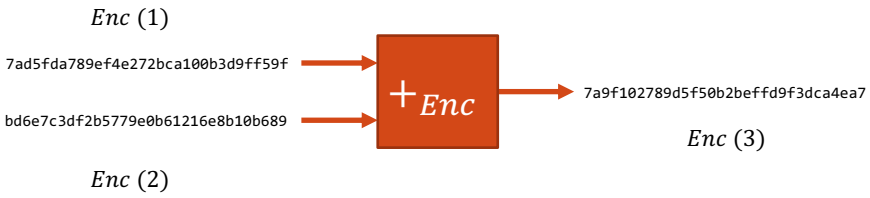
Encryption "hides" data, so traditional query processing does not work in an obvious way

POSSIBLE SOLUTIONS

- Decrypt the data-on-the-fly?
- Two problems:
  - encryption/decryption key is in the cloud (trust in a honest but curious player?)
  - plaintext data is visible at least for sometime
- We would like to develop techniques that allow for ***data to remain encrypted everywhere, including memory at all time!***
- Two fundamental approaches
  - Directly compute over encrypted data
    - **Special homomorphic encryption schemes**
    - Challenge: limited class of computations
    - Challenge: Not composable
  - **Use a "secure" location**
    - Computations on plaintext
    - Challenge: Expensive

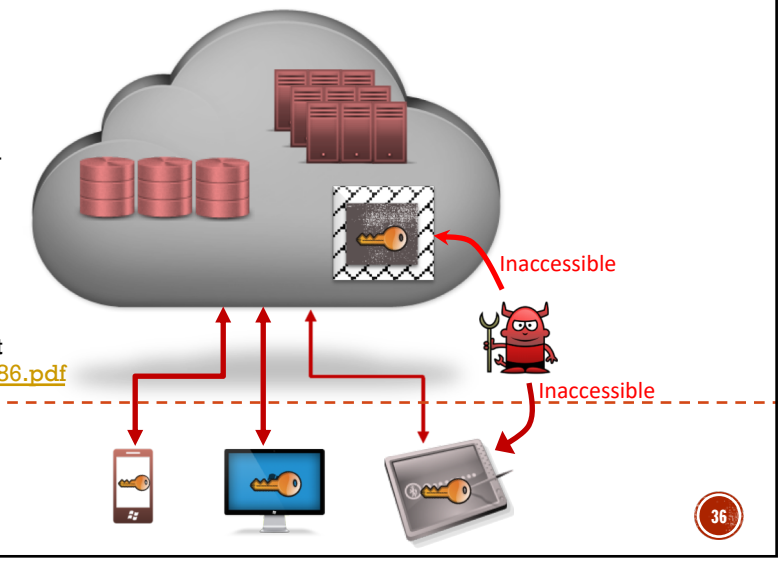
HOMOMORPHIC ENCRYPTION IN A PICTURE

- Directly compute on encrypted values



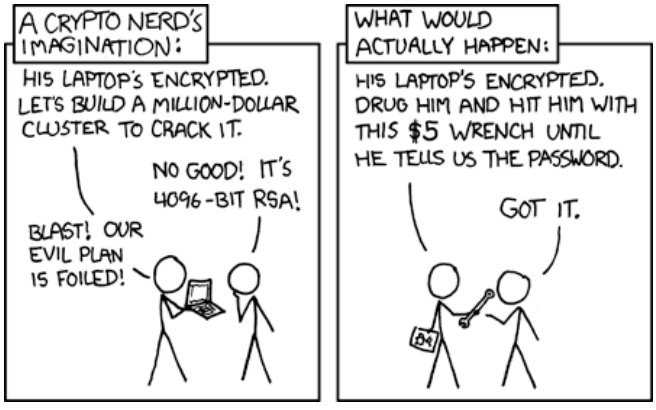
SECURE LOCATION IN A PICTURE

- Computation on isolated components not reachable by attackers
- Example
  - Intel SGX
  - For an overview, have a look at https://eprint.iacr.org/2016/086.pdf



REMEMBER THAT...

- ... there is a lot more to security than encryption



http://xkcd.com/538/

38

ORDER PRESERVING ENCRYPTION (OPE)

OVERVIEW

Range query = asking the server to return ciphertexts in the database whose decryptions fall within a given range, say [a; b]

- OPE is a (deterministic) encryption scheme whose encryption function preserves numerical ordering of the plaintexts
- A DBaas is able to index the (sensitive) data it receives, in encrypted form, in a data structure that permits efficient **range queries**
- Efficient here means in time logarithmic or sub-linear in the size of the database
  - Notice that performing linear work on each query is prohibitively slow in practice for large databases

Value	Enc (Value)
1	0x0001102789d5f50b2beffd9f3dca4ea7
2	0x0065fda789ef4e272bcf102787a93903
3	0x009b5708e13665a7de14d3d824ca9f15
4	0x04e062ff507458f9be50497656ed654c
5	0x08db34fb1f807678d3f833c2194a759e

$$x < y \rightarrow Enc\,(x) < Enc\,(y)$$

OPE: INTUITION (1)

- Generate $|P|$ unique values from a user-specified target distribution and sort them into a table T
- The encrypted value Ci of Pi is then given by $Ci = T[i]$
  - The i-th plaintext value in the sorted list of $|P|$ plaintext values is encrypted into the i-th value in the sorted list of $|P|$ values obtained from the target distribution
- The decryption of Ci requires a lookup into a reverse map
- T is the **encryption key** that must be kept secret

OPE: INTUITION (2)

- The scheme does not reveal any information about the original values apart from the order
  - The encrypted values were generated solely from the user-specified target distribution, without using any information from the original distribution
- Even if an adversary knows all of the encrypted values, it cannot infer T from those values
- Problems
  - The size of the encryption key, namely T, is twice as large as the number of unique values in the database
  - Updates are problematic. When adding a new value P, where $Pi < P < Pi+1$, one needs to re-encrypt all Pj for $j > i$
- OPE is designed in such a way that the result of encryption is statistically indistinguishable from the one obtained using the above scheme, thereby providing the same level of security, while avoiding the two mentioned problems

OPE AT WORK

```
select *
from assignment
where score >= 90
```

$\sigma_{Score \geq 90}$

StudentId	AssignId	Score
1	1	68
1	2	71
3	4	99
...	...	...

```
select *
from assignment
where score_OPE >= 0x04e062ff507458f9be50497656ed654c
```

$\sigma_{Score\_ope \geq 04e0...}$

StudentId	AssignId	Score_OPE
1	1	0x0065fda789ef4e272bcf102787a93903
1	2	0x009b5708e13665a7de14d3d824ca9f15
3	4	0x08db34fb1f807678d3f833c2194a759e
...	...	...

43

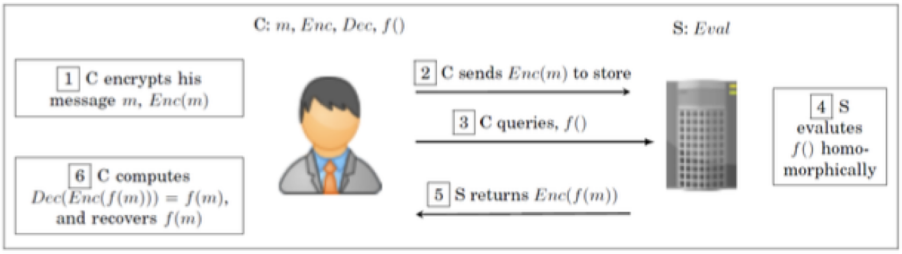
HOMOMORPHIC ENCRYPTION

INTRODUCTION

- Homomorphic encryption is an answer to the question

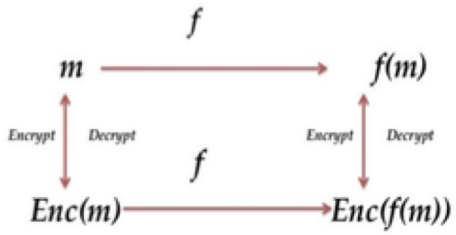
  Can we delegate the processing of data, without giving away access to it?

- Application to cloud computing



DEFINITION

- A homomorphic encryption scheme allows computations on the ciphertext without knowing the secret key, meanwhile ensures that the decryption of the resulting ciphertext is exactly the same as the computations over the plaintext



EXAMPLE APPLICATION: POLLING (1)

- Anonymous polling
  - voting for selecting an option (e.g., on which date and time to held a meeting)
- For simplicity, suppose there are only two options in the poll identified by 0 and 1
- Assume also that there is a host collecting the preferences
- Everyone will submit their choice either 0 or 1 to the host
- The host simply adds everyone's share together and compares whether this final result is larger than half of the population
- Indeed, the polling is not anonymous
  - both the host and whoever can see the polls will know who goes for option 0 or 1
- We want to make it anonymous by using homomorphic encryption

EXAMPLE APPLICATION: POLLING (2)

- First, the host will generate a public (pk) and private (sk) key pair
- Then, the host distributes the encryption key pk to the population
- Every participant encrypts their choice (either 1 or 0) under the encryption key pk and then send the encrypted choice to the host
- The host adds up all the encrypted ciphertexts and forms one ciphertext
- Finally, the host runs the decryption algorithm on the ciphertext obtained in the previous step and derives the result of the poll

PROBLEM AND SOLUTIONS (1)

- It is very difficult to identify encryption schemes that are homomorphic with respect to both addition and multiplication

- For this reason, several different variants of encryption has been designed
  - **Partial homomorphic encryption**
    - Homomorphism property only with respect to some operations and not others
    - Example: if homomorphic with respect to addition (multiplication), it is not possible to compute multiplications (additions, respectively) over ciphertexts
  - **Somewhat homomorphic encryption**
    - The scheme is capable of doing both addition and multiplication to the original plaintexts but its capability is heavily limited
    - Example: it is possible to perform unlimited additions but only one level of multiplication to the ciphertext

PROBLEM AND SOLUTIONS (2)

- **Leveled homomorphic encryption**
  - There is not limit on how we combine ciphertexts together but an upper complexity limit L is imposed on the functionality F that can be computed on ciphertexts
  - If the functionality F can be expressed in a Boolean circuit C such that its depth is lower than the bound L, then it can be evaluated in this scheme

- **Full homomorphic encryption**
  - The scheme is able to perform arbitrary computation to the plaintexts by manipulating the ciphertexts
  - This truly enables **Secure Delegated Computing**
  - If we can find efficient and practical FHE schemes, then we can basically securely offload all of our computations to remote servers without compromising a single bit of data

PARTIALLY HOMOMORPHIC (1)

Homomorphic with respect to **multiplication**: RSA (1978)

- Fix a large number *N=p*q* where *p* and *q* are large primes
- Find pair of numbers (e,d) such that $e * d = 1 \ mod \ \phi(N)$
- Set *(e,N)* to be the public key and *(d,N)* the private key
- $Enc(m) = m^e \ mod \ N$
- Multiplicative Homomorphism property

$$Enc(m_1) * Enc(m_2) = m_1^e * m_2^e \ mod \ N = (m_1 * m_2)^e \ mod \ N$$

  - By simply multiplying two ciphertexts together, one **obtains the encryption of the product of the original plaintexts**

PARTIALLY HOMOMORPHIC (2)

- It is a probabilistic asymmetric algorithm
- Based on the **decisional composite residuosity assumption**, i.e. *given a composite n and an integer z, it is hard to decide whether z is an n-residue modulo n^2*
- *p* and *q* are two large primes
- *gcd(n,(p-1)*(q-1))=1*
- *g* is a random number picked from $\mathbb{Z}_{n^2}$

- Homomorphic with respect to addition: Pallier (1999)
  - Public key: *(n,g)*
  - *n=p*q*
  - $Enc(m) = g^m * r^n \ mod \ n^2$
  - Additive Homomorphism property

$$Enc(m_1) * Enc(m_2) = g^{m_1} * r_1^n \ mod \ n^2 * g^{m_2} * r_2^n \ mod \ n^2 = (g^{m_1} * r_1^n * g^{m_2} * r_2^n) \ mod \ n^2$$
$$= g^{m_1+m_2} * (r_1 * r_2)^n \ mod \ n^2 = Enc(m_1 + m_2)$$

  - The product of two ciphertexts will decrypt to the sum of their corresponding plaintexts
  - Details in the original paper, available at
    https://www.researchgate.net/publication/221348062_Public-Key_Cryptosystems_Based_on_Composite_Degree_Residuosity_Classes

SOMEWHAT HOMOMORPHIC

- A scheme is based on the notion of pairing (see next slide)
- The advantage of pairing is that it allows us to check if the value of *r* is equal to the product of *a* and *b*
- This can be done as follows
  - given *g^r, g^a, and g^b*
  - checking if *e(g^a,g^b)=e(g^r,g^1)* is equivalent to *g^(a*b)=g^r*
- It is possible to combine additive homomorphic scheme with pairing in such a way to compute both the addition and multiplication of the exponents in the group
- However, only a limited number of multiplications can be performed as the group on which pairing is applied may generate a group that does not satisfy the conditions for both the additive homomorphic scheme and the pairing

Let G_1, G_2 be additive groups and G_T a multiplicative group, all of prime order p. Let $P \in G_1, Q \in G_2$ be generators of G_1 and G_2 respectively.

A pairing is a map: $e : G_1 \times G_2 \to G_T$

for which the following holds:

1. Bilinearity: $\forall a, b \in \mathbb{Z} : \ e(aP, bQ) = e(P, Q)^{ab}$
2. Non-degeneracy: $e(P, Q) \neq 1$
3. For practical purposes, e has to be computable in an efficient manner

https://en.wikipedia.org/wiki/Pairing

DIGRESSION ON PAIRING

- in cryptography

54

FULL HOMOMORPHIC ENCRYPTION

INTRODUCTION

- In 2009, Craig Gentry proposed the first fully homomorphic encryption scheme in his PhD thesis *A Fully Homomorphic Encryption Scheme*
- It centers around a function which introduces a certain level of **noise** into the encryption
  - Each operation on the ciphertext results in compounding noise
  - The noise is resolved with the bootstrapability of the encryption
    - Each re-encryption cuts down the noise
    - Analogy to Alice's jewelry shop
- Involves operations on Ideal Lattices
  - Allows for less complex circuit implementation
  - Correspond to the structure of Rings

EXAMPLE (1)

- Take as key a random prime number p
- For a bit m, let $[m] = m \bmod 2$
- Pick a random number q
- $Enc(m, p) = c = [m] + p * q$
  - Notice that $[m]$ can be seen as noise added to the plaintext
- Let $[c]=c \bmod 2$ be the noise associated to the ciphertext
- $Dec(p, c) = [c] \bmod p$
- Homomorphism property

$$Enc(m_1, p) * Enc(m_2, p) = ([m_1] + p * q_1) * ([m_2] + p * q_2)$$

implies

$$Dec\big(Enc(m_1, p) * Enc(m_2, p)\big) = [([m_1] + p * q_1) * ([m_2] + p * q_2)] \bmod p$$
$$= ([m_1] + p * q_1) * ([m_2] + p * q_2) \bmod 2 \bmod p = [m_1] * [m_2] \bmod 2 = m_1 * m_2$$

EXAMPLE (2)

- The compounding noise ($[m1]$ * $[m2]$ in the example) results in loss of homomorphic property after a certain number of operations
  - The bootstrapping of the algorithm allows for this noise to be reduced, allowing for no limit in operations
- However, the combination of the noise production followed by the noise reduction makes the scheme completely impractical
  - Complexity grows as more and more operations are performed (inherent limitation of the algorithm)
  - Gentry stated that in order to perform one search on Google using this encryption, the amount of computations needed would increase by a trillion
  - More schemes have been introduced to try and decrease this complexity, but all rely on the same
- Despite this impracticality, Gentry's discovery is an amazing breakthrough in cryptography and proves that (at least theoretical) fully homomorphic encryption schemes exist

58

MULTI PARTY COMPUTATION FOR HOMOMORPHIC ENCRYPTION

MULTI PARTY COMPUTATION (MPC)

- This stands for a class of problems in Cryptography where we have
  - a group of (possibly malicious) parties, each holding their own private input *Xi*
  - together they wish to evaluate some functionality *F* and obtain *F(X1, …, Xn)*
- Under the requirement that all the parties do not want to reveal their private input to anyone
- Thus, every party should obtain the evaluation result of the functionality *F* and nothing more
- There are several approaches to perform secure multi party computation, the most important ones are
  - Garbled circuits [https://en.wikipedia.org/wiki/Garbled_circuit]
  - Oblivious transfer [https://en.wikipedia.org/wiki/Oblivious_transfer]
- Notice that parties in the protocol perform some computation and exchange information among them, i.e. an active participation is required

APPLICATION OF MPC TO FULLY HOMOMORPHIC ENCRYPTION

- It is possible to solve the FHE problem with any MPC protocol
- Take the user with private input to be one party of the protocol
- Take the delegated server to be the other party
- The user has its private input *PTi*
- The server has its private functionality *F*
- Using an MPC protocol, the user can receive the result without disclosing its input to the server
- Although the FHE problem seems to become trivial when using MPC, there is a serious drawback
  - Since MPC requires interaction between the participating parties, the user has to remain online during the process of the protocol
  - This defies the purpose of FHE, because we would want the user to offload computation to a third party, not participate in computation with it