Database Security

Alessandro Pellegrini
a.pellegrini@ing.uniroma2.it

What is database security?

- Database Security is the set of mechanisms and best practices that protects data stored in the database against *intentional* or *accidental* threats
- A *security policy* describes the security measures enforced
- *Security mechanisms* of the underlying DBMS must be utilized to enforce the policy
- Additional system and management policies external to the DBMS should be enforced to enhance the security level of the data stored in the database

Requirements and Goals

- Security curriculum is relatively light in database systems area
- Focus currently on protecting information through network configuration, systems administration, application security
- Need to specifically consider database system security issues

- Understand security issues in:
  - a general database system environment and a specific DBMS implementation
  - Consider database security issues in context of general security principles and ideas
  - Consider issues relating to both database storage and database system communication with other applications

SQL Injection

```
char stmt_buf[1024];
sprintf(stmt_buf, "SELECT * FROM mytbl WHERE name='%s'", name_val);
```

tale approccio è insicuro

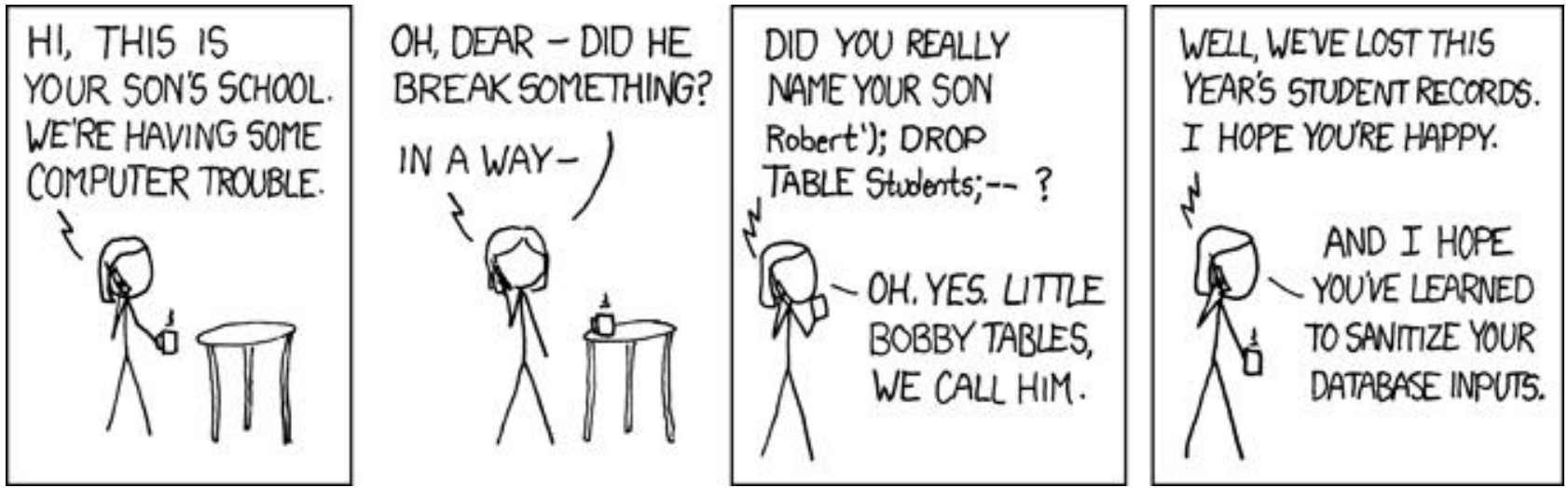
```
SELECT * FROM mytbl WHERE name='O'Malley, Brian'
```

`mysql_real_escape_string()`

caratteri speciali problematici

```
SELECT * FROM mytbl WHERE name='O\'Malley, Brian'
```

SQL Injection



Prepared Statements

meglio lavorare con loro piuttosto che Stored Procedures, difficili da mantenere.

- Prepared statements are a method to allow a DBMS prepare an execution plan of an "incomplete query", where *placeholders* are used in lieu of actual parameters' values
  - `INSERT INTO score (event_id,student_id,score) VALUES(?,?,?)`
- The DBMS analyzes, verifies, and compiles the incomplete query
- Later, when the actual parameters are available, they can be bound to the pre-compiled query
- When the DBMS receives the actual parameters, the actual completed query is executed and the results are sent back
  - This technique was introduced to increase the performance of query execution
  - It can be effectively used to enhance the security of query execution

Database Authentication and Authorization

- Authentication
  - A mechanism that determines whether a user is who they claim to be
  - A system administrator is responsible for allowing users to have access to the system by creating individual user accounts
  - Most modern DBMS systems only allow authenticated users to set up a connection with the DBMS
- Authorization
  - The granting of a privilege that enables an authenticated user to have a legitimate access to a system.
  - They are sometimes referred to as access controls.
  - The process of authorization involves authenticating the user requesting access to objects.
  - This is an important feature of modern DBMS to implement the PoLP

SQL Access Control

- Standard SQL allows to specify who (the *user*) and how (reading, writing, executing, ...) can use a portion of the database
- The object of *privileges* (access grants) are typically tables
  - fine access control is possible: attributes, views, procedures, ...
- A pre-defined user (the *system administrator*) has full privileges by default
  - how this complies with a mandatory access control policy?
- The creator of a resource has full privileges on it

Privileges

- A privilege is characterized by:
  - the referred resource
  - the user granting the privilege
  - the user receiving the privilege
  - the allowed action
  - the capability to transfer the privilege to other users

Privileges offered by SQL

- *insert*: allows to insert new tuples in all or certain tables
- *update*: allows to alter the values associated with already-existing tuples
- *delete*: allows to remove objects from the database
- *select*: allows to read a resource
- *references*: allows to define foreign keys towards a certain resource (this privilege might prevent modification of a resource)
- *usage*: allows to use a resource (for example a domain)
- *execute*: allows to execute a specific stored procedure

grant, revoke, and roles

- Granting privileges:

  grant < *Privileges* | all privileges > on *Resource*
  to *Users* [with grant option]

  - grant option determines whether a certain privilege can be transferred to other users

- Revoking privileges:

  revoke *Privileges* on *Resource* from *Users*
  [restrict | cascade]

- SQL-99 introduces the concept of *role*:
  - a set of privileges
  - a role can be associated with a specific user

Views

tipicamente sono read-only. Normalmente non è possibile apportare modifiche.

- A view is virtual relation that does not actually exit in the database, but is produced upon request by a particular user, at the time of request.
- The view mechanism provides a powerful and flexible security mechanism by hiding parts of the database from certain users.
  - This mechanism provides a finer access control, when coupled with privileges
- The user is not aware of the existence of any attributes or rows that are missing from the view.
- A query can reference a view as if it were a regular table

```
select * from someView
```

- A view can be updated only if defined on a single table (not all DBMS support this operation)

Redundant Array of Independent Disks (RAID)

- Redundant Array of Independent Disks
- The DBMS should continue to operate even if one of the hardware components fails.

- Combine multiple small, inexpensive disk drives into a group to yield performance exceeding that of one large, more expensive drive
- Appear to the computer as a single virtual drive
- Support fault-tolerance by redundantly storing information in various ways
- Uses *Data Striping* to achieve better performance

RAID: Basic Issues

- Two operations performed on a disk:
  - read(): small or large
  - write(): small or large
- Access Concurrency is the number of simultaneous requests that can be served by the disk system
- Throughput is the number of bytes that can be read or written per unit time as seen by one request
- Data Striping: spreading out blocks of each file across multiple disk drives.
  - The stripe size is the same as the block size

minima quantità di byte che posso leggere o scrivere

RAID: Basic Issues

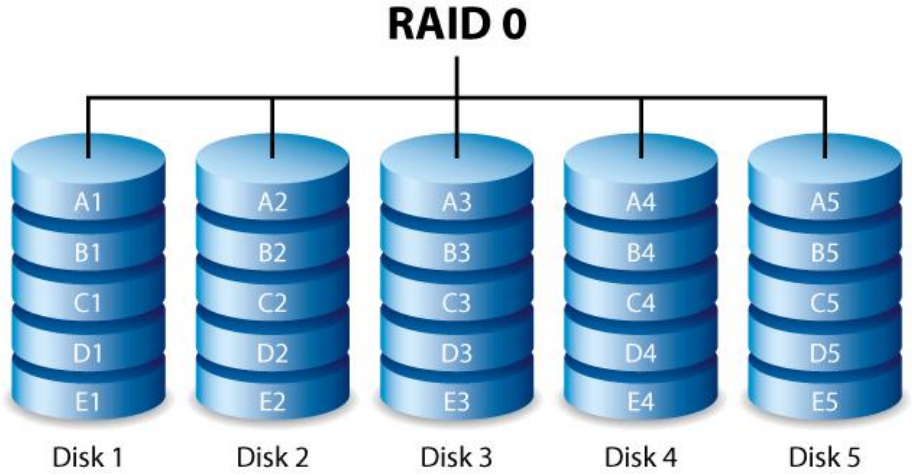
- Striping introduces a tradeoff between I/O throughput and Access concurrency
  - a small stripe means high throughput but limited access concurrency.
  - a large stripe provides better access concurrency but less throughput for single requests

RAID-0

- No redundancy
  - no fault tolerance: if one drive fails, then all data in the array are lost
- High I/O performance
  - parallel I/O
- Best storage efficiency

se perdo A3, nessun altro lo ha!

se devo prendere A2 e A3, lo faccio in parallelo, poichè tra dischi differenti!

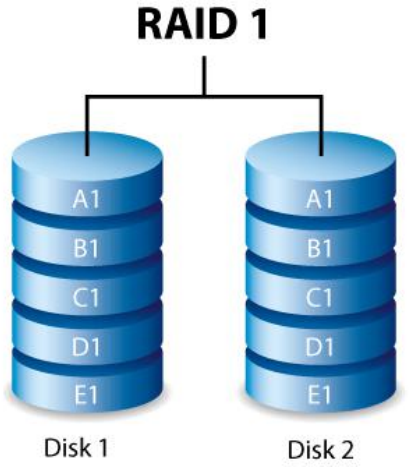


RAID-1

ogni disco ha una copia.
A1 disk 1 e b1 disk 2 possono essere LETTI in parallelo.
se devo SCRIVERE A1, devo farlo su entrambi i disk.

- Disk mirroring
  - poor storage efficiency
- Best read performance: mostly double
- Poor write performance: two disks to be written
- Good fault tolerance: as long as one disk of one pair is working, we can perform R/W operations
- Allows hot pairing

la replicazione è a livello di blocco, r/w su blocchi.

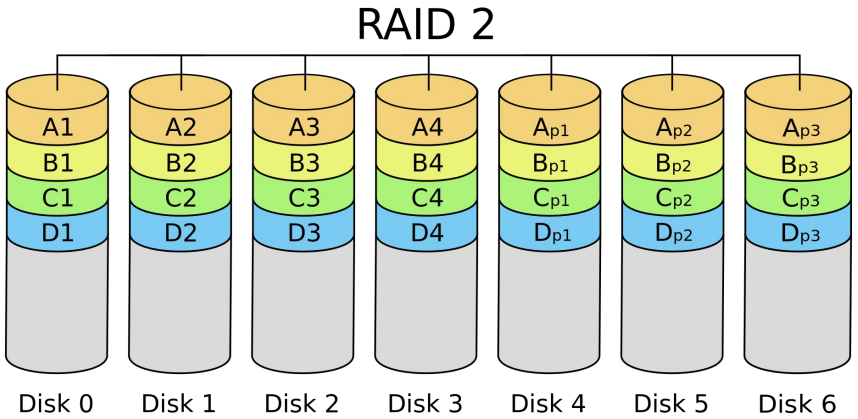


RAID-2

ora memorizziamo bit, non blocchi. Ad esempio, per A_i, vediamo come sia spartito tra i vari dischi. Se perdessi Disk 0, non perderei tutto, perchè possiamo ricostruire il dato intero sfruttando i dischi rimanenti.

- Bit-level striping, rather than block striping
- Uses Hamming codes, a form of Error Correction Code (ECC)
- The disk controller aligns the heads, so that they read the same bits at the same time (no access concurrency)
- Very useful when many read/write errors occur
- Can tolerate one disk failure
- Currently considered obsolete

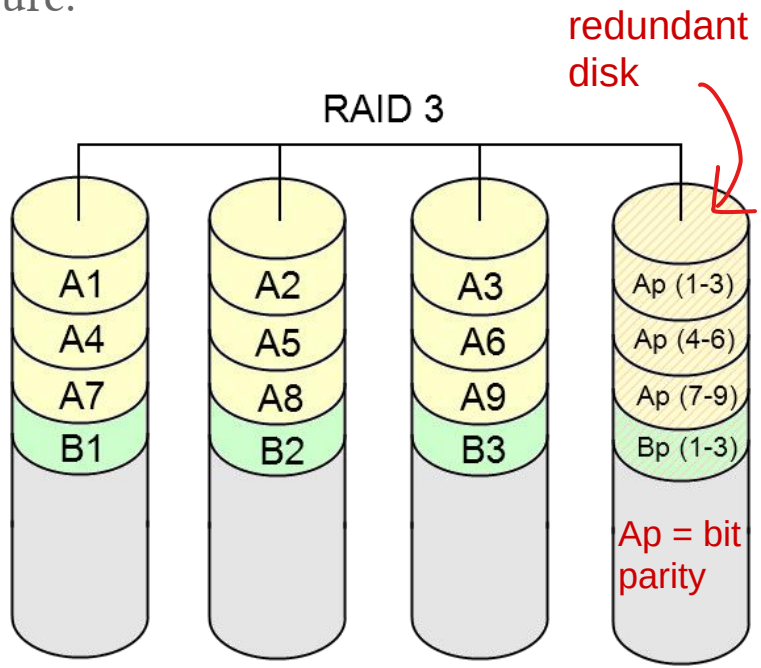
La lettura e la scrittura è in //, perchè
tutte le "teste" dei dischi si muovono all'unisono.



RAID-3

raramente usato

- Byte-level striping, with parity (prima erano bit)
- No need for ECC since the controller knows which disk is in error.
  - parity is enough to tolerate one disk failure.
- Best throughput, but no concurrency.
- Only one Redundant disk is needed.
- Rarely used in practice.



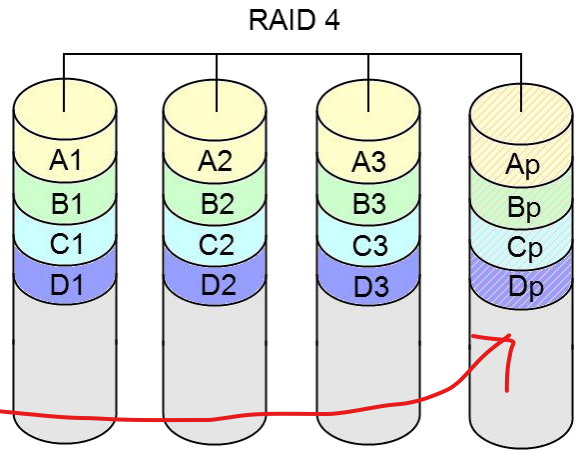
Migliora throughput, perchè la lettura è su un byte, non su un bit.
Se disk fallisce, il controller sa chi ha fallito e può ricostruirlo mediamente il Redundant disk.

RAID-4

possiamo aggiungere altri dischi:
all'inizio non viene usato, poi basta aggiornare il bit di parità dell'ultimo disco.

- Block-level striping
- Stripe size introduces the tradeoff between access concurrency versus throughput.
- Block Interleaved parity.
- Parity disk is a bottleneck in the case of a small write where we have multiple writes at the same time.
- No problems for small or large reads.
- Hot spares can be added

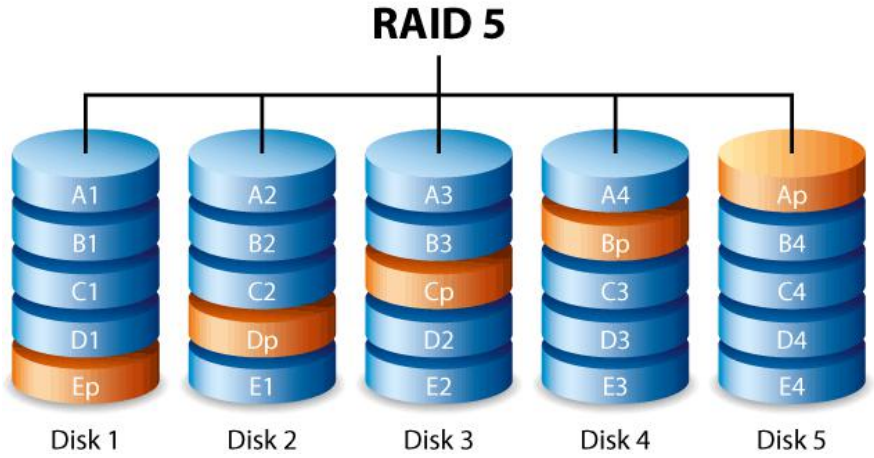
l'ultimo disco è bottleneck
poichè sempre coinvolto!
tutte parity info nello stesso disco,
tutti che ci scrivono



RAID-5

- Block-Level Striping with *distributed parity.*
- Parity is uniformly distributed across disks
- Reduces the parity Bottleneck
- Best small and large read (same as 4)
- Best Large write
- Still costly for small write
- Can tolerate one faulty disk

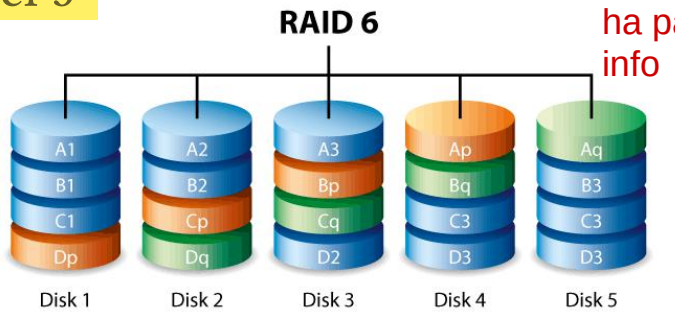
qui i parity information sono tra i vari dischi, quindi è più eterogeneo. Tuttavia, se uno collassa, non ho abbastanza info per usare la parity info.



RAID-6

- Block-level striping with *dual distributed parity*.
  - Two sets of parity are calculated.
- Better fault tolerance: can handle two faulty disks.
- Writes are slightly worse than 5 due to the added overhead of more parity calculations
- May get better read performance than 5 because data and parity is spread into more disks
- If one disks fail, then levels 6 becomes level 5
  - can tolerate two failures

fallimenti indipendenti, quindi possiamo sostituire un disco prima che fallisca. Servono molti dischi.



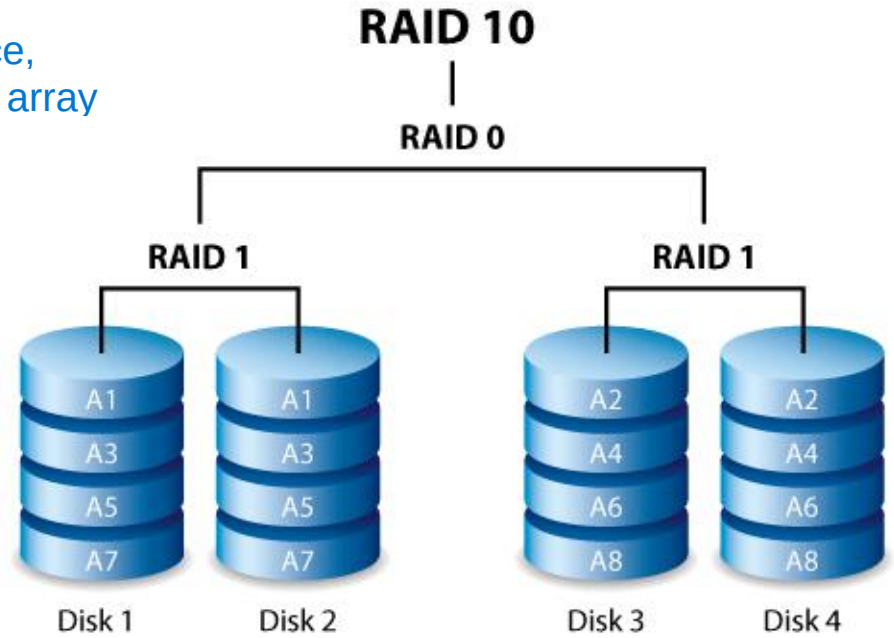
disk 5 ha parity info

RAID-10

per le compagnie

- Merges RAID 0 and RAID 1 in an array
- The ideal choice for RDBMS that need to read/write a large number of small files

RAID0 per le performance,
che però sono in RAID 1 array



Backups

- DBMS should provide backup facilities to assist with the recovery of a database failure.
- Backup and recovery refers to the various strategies and procedures involved in protecting your database against data loss and reconstructing the database after any kind of data loss.
- Backup management should be managed by specific policies
  - where are stored the backups?
  - who can access the backups?

Disaster Recovery

- Disaster Recovery is a set of techniques to protect companies from unpredictable incidents
  - natural disasters, human errors, failures, hacker attacks, ...
- If data are lost, a company incurs significant *losses*
  - A disaster could even bankrupt a company!
- Implementing a Disaster Recovery solutions means
  - replicating company systems and data on another site (at least!)
  - This copy will intervene in the event of a disaster
  - This ensures continuity of operational activities and recovery of systems, data and applications.

Disaster Recovery

- An effective (also cost-wise) disaster recovery entails properly defining two different metrics

- **RPO** (*Recovery Point Objective*): how much data can I afford to lose?
  - This metric indicates the time elapsed from the last data replication, until the disaster occurrence.
- **RTO** (Recovery Time Objective): how long to get back up and running?
  - RTO refers to the time that elapses between the disaster and the complete recovery of the systems.

Encryption

- Data stored in databased are typically unencrypted
  - with small exceptions, *only sometimes*, such as passwords
- If the storage is compromised, all the data are compromised
- Encrypting data in the DBMS can be a viable solution to prevent data losses
  - the cost of transactional accesses might be increased
- Communication channel: it must be also encrypted

- But what if we want to share data with untrusted parties?

Homomorphic Encryption

- Homomorphic encryption enables performing operations on encrypted data without any need for decrypting them
- Beyond classical algorithms for *key generation*, *data encryption*, and *data decryption*, homomorphic schemes offer the following fundamental algorithm:

- *Homomorphic evaluation*: An algorithm **Evaluate** accepts a public key **pk**, a function f and a set of cyphertexts $c_1, \cdots, c_t$ and returns a cyphertext c_f:

$$c_f \leftarrow \textbf{Evaluate}(\textbf{pk}, f, c_1, \cdots, c_t)$$

Correct Homomorphic Encryption

- A Homomorphic cryptographic scheme *HE*={**KeyGen**, **Encrypt**, **Decrypt**, **Evaluate**} is *correct* for a certain function f if the following condition holds:

$$\textbf{Decrypt}(\textbf{sk}, \textbf{Evaluate}(\textbf{pk}, f, c_1, \cdots, c_t)) = f(m_1, \cdots, m_t)$$

- This definition alone cannot guarantee a correct implementation of a homomorphic cryptographic scheme
  - **Evaluate** can be implemented as a simple *identity function*
  - The former condition would therefore hold
  - This is meaningless with respect to computing some function on encrypted data

Somewhat Homomorphic Cryptography

- This is a homomorphic encryption scheme that allows to support addition and multiplication on integers
- The following parameters are considered:
  - η: the length in bits of the secret key
  - γ: the length in bits of the integers in the public key
  - ρ: the length in bits of the *noise*
- The scheme then works as following:
  - the secret key **sk** is an odd integer using η bits chosen in the interval $[2^{\eta-1}, 2^{\eta})$
  - the public key **pk** is $p \cdot q$, where $q \in [0, 2^{\gamma}/p)$
  - **Encrypt**(**pk**, m) returns $c \leftarrow m + p \cdot q + 2r$, where $r \in (-2^{p}, 2^{p})$ is the *noise*
  - **Decrypt**(**sk**, c_i) returns $m_f \leftarrow \left(c_f \bmod p\right) \bmod 2$
  - **Evaluate**(**pk**, $f, c_1, \cdots, c_t$) applies f to the cyphertexts

Somewhat Homomorphic Cryptography

- This is a homomorphic scheme towards addition an multiplication
- Concerning addition:
  - $c_1 + c_2 = (m_1 + m_2) + p(q_1 + q_2) + 2(r_1 + r_2)$, which is the cyphertext of $m_1 + m_2$ but with doubled noise
- Concerning multiplication:
  - $c_1 c_2 = (m_1 m_2) + p(q_1 c_2 + q_2 c_1 - p q_1 q_2) + 2(2 r_1 r_2 + r_1 m_2 + r_2 m_1)$, which is the cyphertext of $m_1 m_2$ but with quadratic noise
- If f is a polynomial of too high degree, the noise will increase to the extent that the cyphertext will become unintelligible with respect to decypher

Leveled Fully Homomorphic Cryptography

- Every somewhat homomorphic cryptography scheme can be transformed in a leveled fully homomorphic scheme
- KeyGen generates a sequence of public/private keys $\langle \mathbf{pk}_1, \cdots, \mathbf{pk}_{k+1} \rangle$ and $\langle \mathbf{sk}_1, \cdots, \mathbf{sk}_{k+1} \rangle$
- Secret keys are then encrypted: $\langle \overline{\mathbf{sk}}_1, \cdots, \overline{\mathbf{sk}}_{k+1} \rangle$, where $\overline{\mathbf{sk}}_i \leftarrow \mathbf{Encrypt}(\mathbf{pk}_{i+1}, \mathbf{sk}_i)$
- Evaluation function can then be used as follows:

$$c_{i+1} \leftarrow \mathbf{Evaluate}(\mathbf{pk}_{i+1}, \mathbf{Decrypt}, \overline{\mathbf{sk}}_i, c_i)$$

- The homomorphic scheme is therefore organized as a "circuit", in which every gate initially performs a decryption operation, and then applies f
- The noise is therefore removed and recomputed at every level of the circuit

Private Set Intersection (PSI)

- PSI is a computational problem in which two (or more) parties own a private set of elements
- They are interested in calculating its intersection, without revealing any element (or in general any additional information) of their own dataset outside the intersection itself.

- It is a form of *secure multi-party computation*

Homomorphic PSI

- The receiver has the set S of size N_S, the sender provides the set S' of size $N_{S'}$
  - Both sets are composed of bitstrings of length σ
  - Values N_S, $N_{S'}$, and σ are public
- The receiver can compute $I = S \cap S'$, the sender does not get any additional information

1. **Setup**: sender and receiver agree on a fully homomorphic cryptographic scheme; the receiver generates a keypair, keeping secret the private one
2. **Element encryption**: the receiver encrypts all the elements $s \in S$ and sends the N_S cyphertexts $(c_1, \cdots, c_{N_S})$ to the sender

Homomorphic PSI

3. **Intersection computation**: for each c_i, the sender:
   - generates a non-zero random value r_i
   - homomoprhically computes:

$$d_i = r_i \prod_{s' \in S'} (c_i - s)$$

   - cyphertexts $(d_1, \cdots, d_{N_{s'}})$ are returned to the receiver
4. **Response extraction**: the receiver decyphers $(d_1, \cdots, d_{N_{s'}})$ and computes:

$$I = S \cap S' = \{s_i : \mathbf{Decrypt}(\mathbf{sk}, d_i) = 0\}$$

- Privacy is therefore ensured because elements not belonging to the intersection are mapped to a uniformly-at-random value.