In simple terms, the paper discusses how to protect information stored in computer systems from unauthorized use or modification. It covers different mechanisms, both hardware and software, needed for information protection. The paper has three main sections:

i. **Basic Principles of Information Protection:**

   Introduces the need for protecting information in computer systems. Gives examples of scenarios where different users should have different levels of access to data. Defines terms like privacy, security, and protection in the context of computer systems.

ii. **Descriptor-Based Protection Systems:**
   Goes into more detail about protection architectures, access control, and authentication mechanisms. Discusses different levels of functional capability for information protection. Describes various protection schemes and their functional properties, ranging from unprotected systems to systems that control sharing based on user-defined rules.

iii. **The State of the Art:**
   Reviews the current state of information protection and ongoing research projects. Provides suggestions for further reading on the topic. The paper also includes a glossary of terms related to information protection in computers. Overall, it's a tutorial exploring the mechanics and principles of safeguarding computer-stored information.

The text discusses computer protection mechanisms, focusing on internal mechanisms rather than external ones like contracts or physical barriers. It introduces eight design principles for protection mechanisms:

i. **Economy of Mechanism**: Keep the design simple and small to facilitate error detection.
ii. **Fail-Safe Defaults**: Base access decisions on permission rather than exclusion, with the default being lack of access.
iii. **Complete Mediation**: Every access to every object must be checked for authority.
iv. **Open Design**: The design should not be secret, relying on specific keys or passwords rather than secrecy.
v. **Separation of Privilege**: Systems should require two keys for access, providing more robust security.
vi. **Least Privilege**: Programs and users should operate with the minimum set of privileges necessary.
vii. **Least Common Mechanism**: Minimize shared mechanisms to reduce potential security compromises.
viii. **Psychological Acceptability**: Design the human interface for ease of use to ensure correct application of protection mechanisms.

The text acknowledges the difficulty of creating flawless systems and suggests relying on these principles to mitigate potential issues. It also briefly discusses two additional design principles proposed by traditional security analysts: work factor (comparing the cost of circumventing a mechanism with the attacker's resources) and compromise recording (reliable recording of information compromise).

The text then transitions to the technical underpinnings of information protection in computer systems. It presents a bottom-up approach, introducing models of systems as they are or could be built. The first model describes a multiuser system that isolates users from each other, and it introduces the concept of a descriptor register for memory access control. This register is controlled by a privileged state bit, and a supervisor program manages the virtual processors in the system.

The authors emphasize direct access to information, discussing the essentials of information protection. The concept of descriptors is introduced, representing objects to be protected, and the text explains how a descriptor register controls memory access. The privileged state bit is crucial for protection, and a supervisor program with this bit ON manages the simulation of virtual processors. The text concludes by noting that protection mechanisms not only safeguard users from each other but also protect their own implementation.

The virtual processor implementation discussed in the provided text involves three protection mechanisms associated with abstractions:

i. **Memory Access Protection:**

The first mechanism protects distinct programs by enforcing descriptor restrictions. A guard, represented by additional hardware, enforces these restrictions, allowing access to memory only through the descriptor mechanism.

An authority check ensures that memory access requests lie within the specified area of memory based on the virtual processor's identification using base and bound values in the descriptor register.

ii. Descriptor Register Protection:

The second mechanism safeguards the contents of the descriptor register. The guard, implemented in hardware, allows loading the descriptor register only if the privileged state bit is ON (supervisor program). It restricts loading for user programs when the bit is OFF.

iii. **Privileged State Bit Protection:**

The third mechanism protects the privileged state bit, allowing transitions between user and supervisor modes. User programs (privileged state bit OFF) can turn the bit ON and transfer to the supervisor program, while supervisor programs (privileged state bit ON) can turn it OFF.

The supervisor program is crucial for maintaining the integrity of base, bound values, and the privileged state bit, preventing incorrect labeling of virtual processors.

Additionally, the text introduces the concept of a virtual machine, which involves combining a virtual processor, a memory area, data streams, and an isolated region of long-term storage. Long-term storage is accessed less frequently and is often controlled by supervisor programs.

The text then delves into authentication mechanisms, particularly in time-sharing systems, where user identity verification relies on secrecy, often involving passwords. The weaknesses of password-based systems are discussed, leading to considerations of unforgeable objects like keys or cards for authentication.

Furthermore, the text explores the need for sharing information among users. It introduces two general implementations for protection mechanisms that permit sharing: list-oriented and ticket-oriented. The concept of principals, representing entities accountable for virtual processor activities, is introduced. The role of authentication mechanisms in associating virtual processors with principals is highlighted.

Finally, the text discusses the implications of sharing, including access restrictions to shared areas, careful handling of shared resources, and the need for the supervisor to manage authorized principals for shared resources. The text concludes by emphasizing the importance of controlling conditions for virtual processors to enter a domain, ensuring proper accountability.

The implications of sharing a procedure among multiple virtual processors:

Temporary Result Storage:

Shared procedures need to be cautious about where they store temporary results, especially when used concurrently by multiple virtual processors. Enforcing access permissions via descriptor bits adds constraints, preventing a shared procedure from modifying itself. For instance, if program A is restricted from writing into the shared math routine, the routine is also prohibited from writing into itself.

Expansion and Generalization:

The need for expansion to accommodate several distinct shared items can be addressed by increasing the number of descriptor registers. Two general forms of this expansion are capability systems (ticket-oriented) and access control list systems (list-oriented).Most real systems use a combination of both forms, employing capability systems for speed and access control list systems for the human interface.

Capability Systems vs. Access Control List Systems:

Capability systems use encapsulated, copyable descriptors as tickets for flexible authorization. Access control list systems involve lists of authorized users associated with shared objects. The text suggests that real systems often integrate both capability and access control list systems.

Authorization and Development of Protection Models:

The text proposes the need for more groundwork before delving into generalizations, including authorization.

Section II of the text promises the development of progressively more sophisticated protection models. Capability systems are explored in this context, emphasizing the use of encapsulated descriptors as flexible tickets for authorization.

Communication External to the Computer:

The general rule is established that external communication must precede dynamic authorization of sharing. Copyable descriptors, while flexible, lack accountability for their use, leading to the need for analysis of revocation and the introduction of indirection.

Access Control Lists and Authorization Control:

Access control lists embedded in indirect objects are introduced for detailed control of authorization. The discussion extends to controlling changes in authorizations, considering different models with varying susceptibility to abuse.

Sensitive Data Release and Constraints:

Releasing sensitive data to a borrowed program requires additional control of authorization changes. A nonintuitive constraint on where data may be written by the borrowed program is implied.

Arbitrary Abstractions Implementation:

Section II explores the concept of implementing arbitrary abstractions, such as extended types of objects, as programs in separate domains. In this section on Descriptor-Based Protection Systems, the text discusses the separation of addressing and protection, primarily focusing on the capability system within descriptor-based protection.

Separation of Addressing and Protection:

Descriptors are introduced for protecting information, and they may also be used for organizing addressing and storage allocation. The text proposes the separation of organizational and protective uses of descriptors by requiring all memory accesses to go through two levels of descriptors. This separation is conceptually achieved by enlarging the memory system to provide uniquely identified storage areas called segments, each with a distinct addressing descriptor. The addressing descriptors consist only of a base and a bound value, serving both organizational and protection purposes.

Unique Identifiers and Segments:

Unique identifiers (such as a hardware counter register) are crucial for identifying segments and are used by the protection system. The processor has protection descriptors containing unique segment identifiers, separating the system address space (universal) from the processor address space (local). Different virtual processors have different processor address spaces, but the system address space remains the same for all users.

Capability System:

The text introduces the capability system, a generalization that allows users to place protection descriptor values in memory addresses for convenient access. Capabilities, memory words containing protection descriptor values, are a key concept in the capability system. The text illustrates the capability system with an example involving protection descriptor registers, capabilities, and addressing segments.

Dynamic Authorization and Authentication:

The dynamic authorization of sharing is explained in the context of the capability system. The protocol for dynamically authorizing sharing involves communication outside the computer system to securely exchange principal identifiers.

Revocation and Control of Propagation:

The capability system's efficiency, simplicity, and flexibility are highlighted, but potential issues like revocation, propagation control, and access review are discussed. Problems like revocation and uncontrolled propagation are addressed, and potential solutions such as adding a copy bit to capabilities are mentioned.

Constraints on Capabilities:

Constraints on capabilities, like the use of a copy bit or designating specific capability-holding segments, are proposed to mitigate potential issues. The text explores approaches to limit the effort of auditing and enhance revocation possibilities.

Depth Counters:

Another approach involves associating a depth counter with each protection descriptor register to limit the length of the chain by which a capability may propagate. Depth counters are suggested to reduce the effort required for auditing and to limit the potential accessors in the system.

In summary, this section provides a comprehensive overview of the capability system within descriptor-based protection systems, covering the separation of addressing and protection, unique identifiers for segments, dynamic authorization, and potential issues and solutions related to revocation and control of propagation.

The text continues to discuss the extension of the basic capability mechanism to enhance control over revocation. Redell's proposal involves introducing indirect objects that force capabilities to specify their targets indirectly through a second location before reaching the actual object of interest. These second locations, called indirect objects, are independently addressable recognizable objects. Anyone with the appropriate capability for the indirect object can destroy it, thereby revoking access for others who have a capability for that indirect object.

Redell's indirect objects are compared to the access controllers of the access control list (ACL) system. While they provide a systematic revocation strategy, they offer limited assistance for issues like propagation and auditing. The fundamental problem is that authorizations, or bindings, occur whenever a capability is copied, and there is no provision for reversing this binding unless an indirect object is created for the copy.

The passage also explores the challenges of limiting copyability, which can help control authorizations but hinder the simplicity, flexibility, and uniformity of capabilities. The dilemma arises as restrictions on copyability impact the usefulness of capabilities in the context of procedure calls, hindering good programming practices.

The Access Control List (ACL) system is introduced as a strategy to provide reversibility of bindings by inserting an extra authorization check at the latest possible point. The ACL system delays authorization until the last possible moment, unlike the capability system, which is ticket-oriented.

The ACL system involves creating access controllers, which are special objects containing an addressing descriptor for the associated segment and an access control list. The ACL system introduces protection groups, allowing principals to be part of a group for access authorization. Various implementation considerations are discussed, including the use of shadow registers to optimize memory access and potential challenges in implementing access control lists.

The passage also touches upon the authority to change access control lists, introducing concepts of self-control and hierarchical control. The hierarchical control scheme involves a hierarchy of access controllers, and the prescript field is introduced to trigger actions based on changes to access control lists.

The passage continues to discuss **discretionary and nondiscretionary controls**. Discretionary controls allow users to determine access authorization at their discretion, while nondiscretionary controls are imposed externally. The need for nondiscretionary controls is highlighted, especially when dealing with complex programs from external suppliers to prevent security violations. The idea of confining borrowed programs to specific domains, limiting their ability to share information, is introduced as a way to achieve security in the presence of borrowed programs.

The passage discusses challenges and strategies related to implementing discretionary and nondiscretionary controls in computer systems. The coexistence of these controls is considered difficult, with suggestions for implementing nondiscretionary controls using a second access control list system in parallel with discretionary controls. The focus is on achieving compartmentalization, where data objects are labeled for specific compartments, and access is granted only if the compartments align.

The discussion delves into the complexities of implementing nondiscretionary controls, particularly in scenarios where a principal is authorized to access multiple compartments simultaneously. The need for strict controls on reading and writing is emphasized to prevent potential security breaches, such as the compromise of compartment boundaries.

The concept of "**high water mark**" and the strategy proposed by Bell and LaPadula are mentioned as approaches to address the issue of writing into objects with insufficient compartment labels. The need for human judgment in the declassification of sensitive information is acknowledged, and the passage suggests that developing systematic ways to incorporate human judgments is a research topic.

The text also touches upon the challenges of achieving complete confinement of a program in a shared system, highlighting the potential for subtle strategies, such as varying paging rates, that could signal information to other users. The term "banging on the walls" is introduced to describe this problem.

The discussion extends to the protection of objects other than segments, introducing the concept of type in the protection system. The passage illustrates how various system-implemented objects, such as data segments, access controllers, FIFO message queues, and input/output streams, can be protected with different operations permitted based on their types.

The notion of **protected subsystems** is introduced as a solution to overcome limitations in controlled sharing. Protected subsystems involve encapsulating program and data segments to prevent unauthorized access, allowing arbitrary controls on sharing. The passage emphasizes that each protected subsystem operates in its own domain, and mechanisms for associating multiple domains with a computation are discussed.

This section concludes by briefly mentioning the challenges and considerations in implementing domain switching, emphasizing the coordination of protection domains with dynamic activation records, variable storage, and controlled passing of arguments between domains. The principles of "**separation of privilege**" and the need for careful coordination in achieving protected objects and subsystems are highlighted.