Why should passwords not be stored in plain text in a database?



www.yzokul.com.tr

Why should passwords not be stored in plain text in a database?



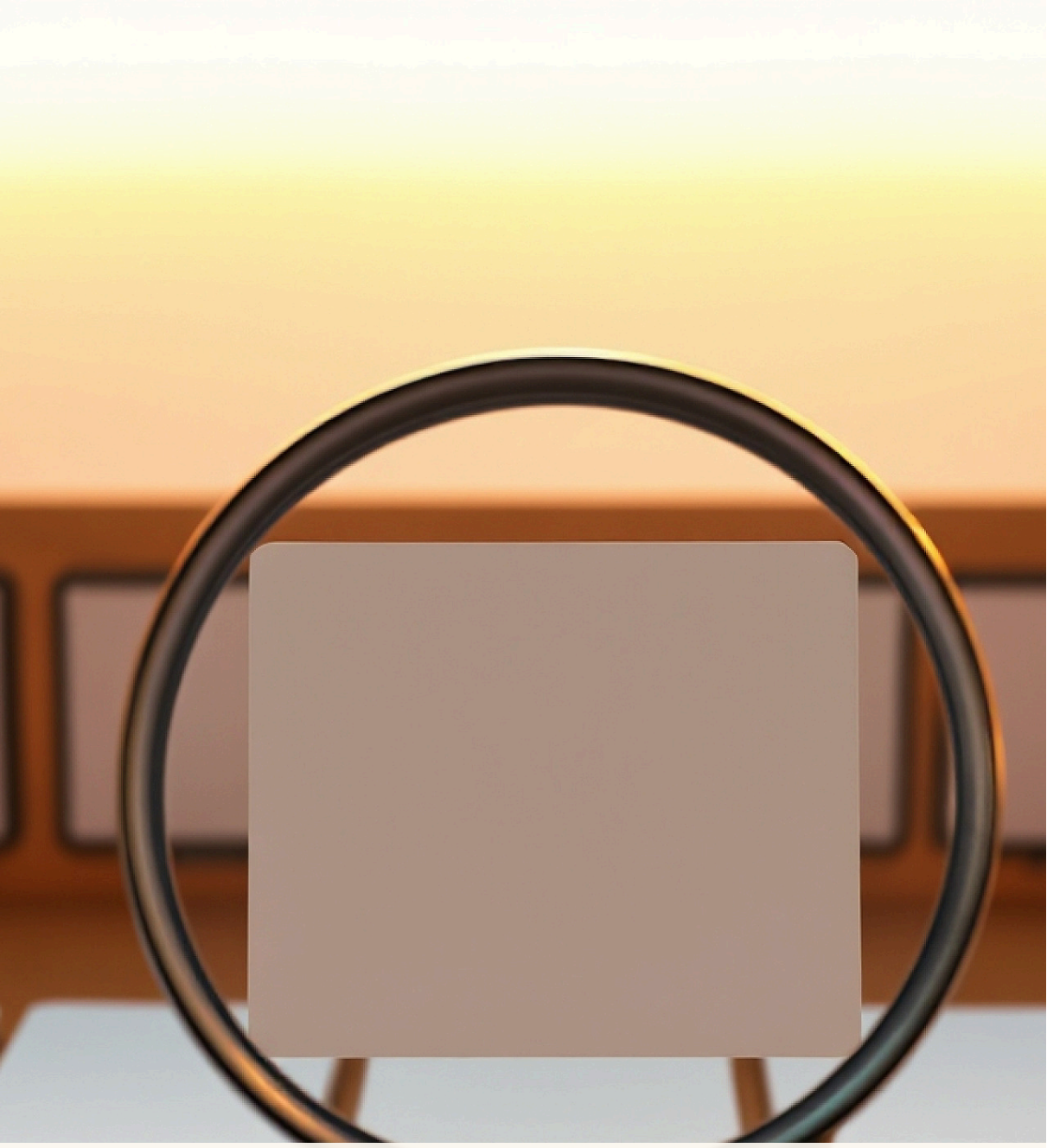
You see the question: why should passwords not be stored in plain text in a database? The short answer is that anyone who captures the records could simply read the passwords. That means every stored password would be exposed the moment the records are taken. So we must protect passwords instead of leaving them readable. This book explains how that protection works, step by step, for complete beginners.



You understand that a database is a collection of records kept by a system, such as user accounts. Each record can hold fields, which are labeled pieces of data like a username and a password. The database is the place where the system keeps this information so it can check it later. When you create an account, a new record is added with your details. The password field is the part we care about most here.



You see that plain text means the password is written exactly as the user typed it, with no change at all. Storing it this way means the record contains the readable password itself, letter for letter. There is no scrambling, no encoding, and no transformation of any kind. Anyone who looks at the record sees the password just as the user would type it. That is the simplest possible way to store it, and the most dangerous.



You picture someone who captures the records opening them and reading the passwords directly. No special skill is needed because the passwords are already readable text. They do not have to guess, decode, or use any tool; they simply look. This is the plain text risk: the secret is sitting in the open. Even a beginner with the records could read every password in seconds.



You understand that a single capture of the records exposes every stored password at once. One leak hits everyone whose account is in that database, not just one person. Each affected user's account is now open to misuse by whoever holds the records. The damage spreads across all users in a single event. That is why the way passwords are stored matters so much for everyone, not only for one account.



You see that many people reuse passwords, using the same one on other services too. A readable password from one system can therefore put accounts elsewhere at risk. Someone who reads it here may try it on email, social media, or shopping sites. The harm does not stop at the system that was captured. Password security on one site matters for the user's other accounts as well.



You understand the aim: the system should not need to read back the original password at all. It only needs to check whether the entered password matches the one that was set. The system never has to display the password or send it back to anyone. This changes the design: store something that lets you check, not something you can read. That idea is the key to safer storage.



You see that a hashing method turns the password into a fixed-looking value through a one-way process. From that value, the original password is not meant to be recovered. The same input always gives the same output with the same method, which is what makes checking possible. Hashing is not encryption, because there is no key to turn it back. It is a fingerprint of the password, not the password itself.



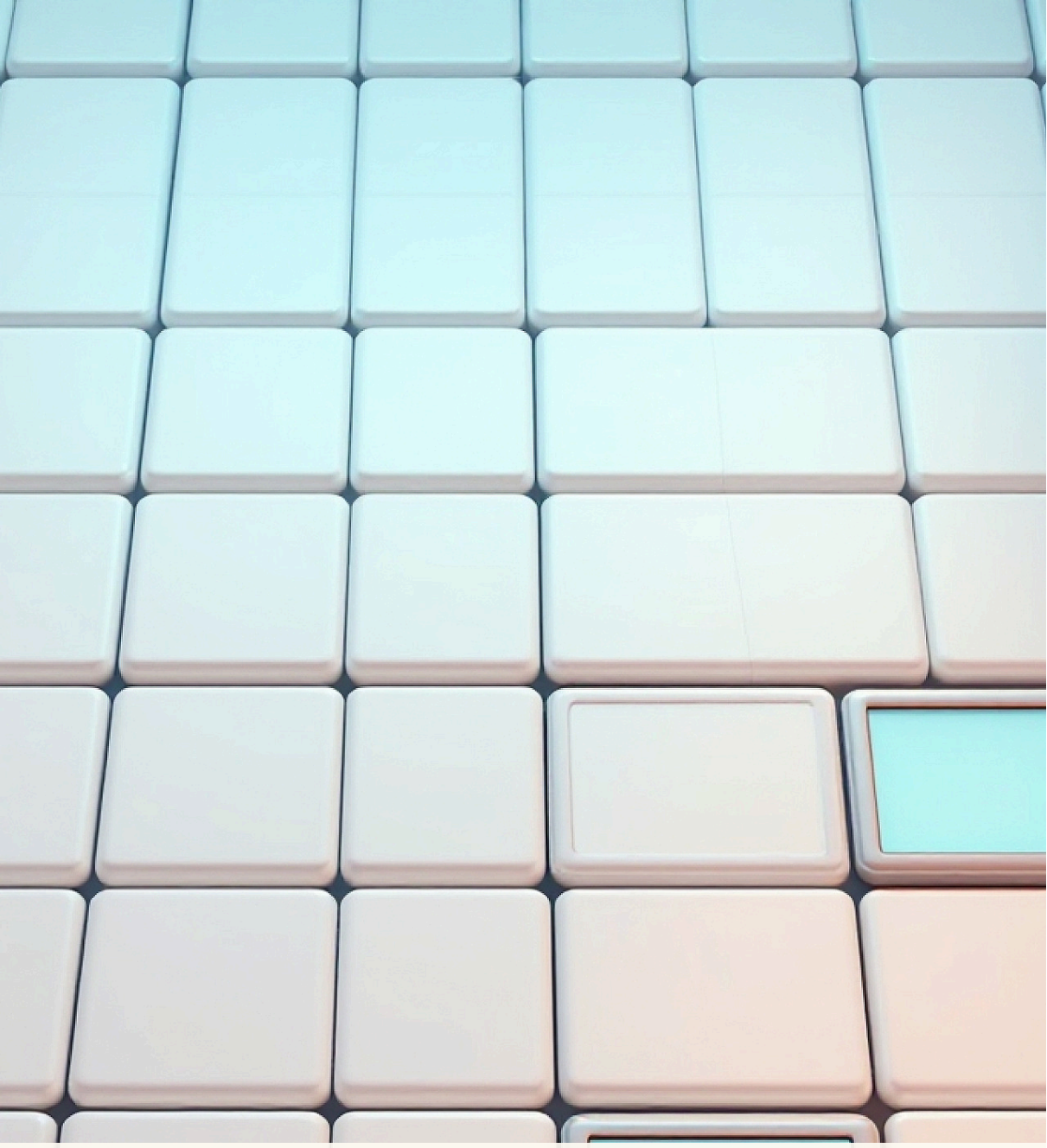
You picture the database record now holding the hashed value instead of the readable password. Someone who captures the records sees only the hash, not the password. The hash looks like a jumble of characters with no obvious meaning. Reading it does not tell the attacker what the user typed. This is the core of password hashing: store the processed value, never the original.



You follow a login: the user types a password, the system processes it with the same hashing method, and compares the result with the stored value. A match means the password was correct, and a mismatch means it was wrong. The system never reads back the original password; it only compares processed values. This is how a system checks a password without ever storing it in readable form.



You understand that a salt is extra data mixed in before hashing. It changes the resulting value even when two users choose the same password. The salt is stored alongside the hash so the system can repeat the same process at login. It does not need to be secret; its job is to make each result different. Adding a salt is a simple step that greatly improves protection.



You see that each record gets its own different salt value. Identical passwords then produce different stored values, so one guess does not unlock many records. Without this, two users with the same password would have identical hashes, and an attacker could spot them instantly. A different salt per record breaks that pattern. This is why salt values should never be shared across records.



You learn that a simple and fast hash may not provide sufficient protection on its own. Fast processing lets an attacker try many guesses quickly against a captured hash. Speed is useful for the system, but it also helps the attacker. So password hashing should use methods designed to be slow and costly to guess. Fast hashes alone are not enough for password security.



You spot the common mistake of keeping passwords readable, perhaps thinking the database is private enough. The records can still be captured, and then the passwords are exposed. Privacy settings do not change the fact that the stored text is readable. Assuming the database is safe is not a protection. The fix is to stop storing readable passwords and hash them instead.



You spot the mistake of hashing without a different salt for each record. Equal passwords then look equal in storage, weakening the protection. An attacker who captures the records can group identical hashes and target them together. One guess can then reveal many accounts at once. Always use a different salt value per record to avoid this weakness.



You sum up: never store passwords in plain text, because captured records would be readable. Instead hash with a different salt per record, compare the processed input at login, and remember that simple fast hashes may not protect enough alone. A single leak would otherwise expose every user at once. Follow this path, and the system never needs to read back the original password.