All this text wasn’t written by me (Tiago a.k.a TigaxMT).

All of it was transcribed by me from HackerOne video lessons.

All credits and thanks go to them.

Secure Password Storage

USE BCRYPT

That's pretty much it. When you want to store password on the server, use the BCrypt package for your particular language.

It's an extremely resilient password digest solution, and solves the problem effectively.

GOALS

Our goals for secure password storage are:

- Impervious to rainbow tables (huge tables of per-hashed passwords, that you can compare to extracted hashes)
- Computationally expensive, so brute-forcing is impractical and difficult
- Unique per-user, so that cracking one hash can't give you the password of multiple users

SAY NO TO MD5

MD5 is commonly said to be "broken". What that means is that there are theoretical attacks that can weaken it. But none of them matter for passwords.
What matters is that MD5 – and the SHA family of hashes – is really fast. Which is great, if you're attempting to validate that files haven't been tampered with. But it means that brute-forcing is fast too!

If MD5 and friends aren't acceptable, what do we use instead?
The PBKDF1 and 2 constructions perform thousands of rounds of these simple hashes, to iteratively build up the returned hash.
This slows them down to the point where comparing a single password hash could take hundreds of milliseconds. More rounds, more time.

SALTING

Salting is the process of adding a random value to the beginning and/or end of a password, to make rainbow tables useless.

When you do this, though, it's important to not have a single global salt value for every hash. That would violate our goal of user hashes being unique, even if they're using the same password.

USE BCRYPT

The reason I said initially to use BCrypt is that it really solves all of these out of the box, greatly diminishing the chances of you getting it wrong.
SCrypt is a less battle-tested but still great solution. It’s designed to not only be computationally expensive, but to use lots of RAM, making it hard to parallelize.

IF NOT BCRYPT

If neither of these are an option, though, my recommendation is this: SHA256 in PKDF2 using per-user salt values (randomly generated when the password changes), with at least 10000 rounds.
This will be resilient to the vast majority of attacks for years to come, and you can always increase the round count later.