# Passwords stored using reversible encryption: how it works (part 1)

Archived: 2026-04-06 02:08:29 UTC

In case you missed my HAR2009 talk: in the second part I talked about a Windows feature called ‘Store passwords using reversible encryption’. When this is enabled (per user or for the entire domain), Windows stores the password encrypted, but in such a way that it can reverse the encryption and recover the plaintext password. This feature exists because some authentication protocols require the plaintext password to function correctly, the two most common examples are HTTP Digest Authentication and CHAP.

This feature is not enabled by default but I’ve seen it a couple of times in customer networks. As I couldn’t find any description of how this mechanism works or any tool to recover these passwords, I decided to investigate.

When you change your password on a domain that has reversible encryption enabled, a password filter called RASSFM.DLL is used to store the password using reversible encryption. The key that is used to do this is G$MSRADIUSCHAPKEY, which is stored as a global LSA secret. This key is decrypted using a static key (hardcoded in the DLL). The result of this operation is combined with a 16-byte random value (generated every time someone changes their password) and that key is used to encrypt a Unicode version of the password using the RC4 algorithm.

I found out these passwords are stored in Active Directory in a per-user structure called userParameters. If you use a tool such as AD Explorer you can look at this structure in an AD that has enabled this feature. When you look at this structure, it looks like a binary blob, with some human-readable parts in there. When you enable reversible encryption you will notice two readable strings: G$RADIUSCHAP and G$RADIUSCHAPKEY. The userParameters can also be used to store settings unrelated to reversible encryption, such as per-user Terminal Server settings.

Following the G$RADIUSCHAP part is the ascii-hex encoded encrypted password. The part following the G$RADIUSCHAPKEY name is the 16-byte random value.

So to decrypt this password we use the following steps:
- Take the G$MSRADIUSCHAPKEY Global LSA secret
- Decrypt it using the static key
- Parse the userParameters structure and extract the G$RADIUSCHAP and G$RADIUSCHAPKEY values
- Combine the value of G$RADIUSCHAPKEY (the 16-byte random) with the decrypted LSA secret to create an RC4 key
- Decrypt the value of G$RADIUSCHAP using that RC4 key

The result is a plaintext Unicode password. My tool 'Revdump' automates this procedure.

In part two of this article, I will look at the security of this mechanism.

---