# My 4D Login Dialog Replacement

## by Dani Beaubien

## Introduction

In light of the recent discussions about limited security of the built-in user password system that ACI currently provides, I am sharing a replacement user system that I have been using over the last year and half with great success.

Initially, my reason for building my own login system as shown in figure 1, was that user and password information were required to propagate through a number of distributed databases.The users could not be kept in the structure since there were numerous copies of the structure and there was no way to have a password change propagate properly unless the user information (and password) were stored in the data file directly. At the time that I needed the solution, the user commands that v6 provided simply didn't work as they were documented.

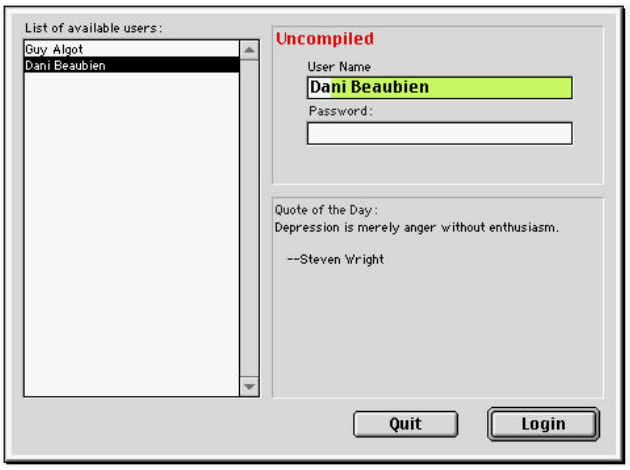


**Figure 1**

I knew that all text data that is stored in the data file is not encrypted. So I used a little inspiration from digital signatures to help me "encrypt" the passwords. When a digital signature is used to "sign" a document you apply a complicated algorithm to the document data to produce a "digest." The resulting digest is basically a very complicate checksum. That digest is stored with the document data. To verify if the document has been tampered with, you simply rerun the algorithm and if the resulting digest matches the stored digest then the document data is unaltered. This is an extremely simplistic description, but it is enough for this article.

## On the examples disk

You'll find a Macintosh 4D example database to accompany this article on this issue's examples disk in the folder named "*Login*." I have purposely not shown the code in this article, because it is best to see it in action. Take a look at the sample database and see it for yourself.

## Passwords are not stored

Now the algorithm that I use is considerably simpler. The method that generates the digest is called *Password_GenerateDigest*. I pass two parameters, the User ID of the user and the typed password. The digest is stored in the user record which can be compared to later for verification when a user is trying to log in. Since the digest is stored, there is no password stored. If you want to verify the password that a user has typed, then run the algorithm and pass in that users' UserID and the password that they typed. If it matches the one stored in their user record then they typed the same password.

The real power of this method is that I can change the algorithm that is used to create these digests as I see fit. If someone were to convert the MD5 algorithm (used in APOP implementations) into a 4D method, you could use that. I could write a digesting algorithm that takes a string of the user ID and the password and does a simple bit shift to get the digest. The more secure you want the password, the more robust and potentially complicated an algorithm you put in the *Password_GenerateDigest* method.

One of the best tricks, when developing an algorithm, is to write the method so that it runs through a number of recursions, using the results of the previous recursion to feed the next. At each level of recursion you want to "throw" part of the result way. So if you return a 80 character result, throw away the first 20 characters and feed the shorten result back into the algorithm. This will make it harder for the resulting digest to be reverse engineered into the originally entered password.

Please remember that the digest is only as secure as the algorithm that you pass the original password through. The algorithm that I have provided is something that I whipped up to demonstrate the concept and should not be considered exceptionally secure. In other words, use this one at your own risk. There a host of encryption books that can provide robust and safe algorithms for encrypting the password to create the digests.

## Try it out

There are essentially two parts, one is a user table and the interface that goes with it. The other is essentially a login dialog. I have placed all the necessary code to test the login dia-

log in the *000_RUN_ME_FIRST* method. When you open the sample database, just enter the user environment and run that method. You will be presented with a login dialog with two users. My password is blank and Guy's password is "test me." Check out the users table for how I manage the passwords and the user information.

To edit the user information, switch to the user table in the user environment.

## Extras

You will quickly be able to tell that there are some fields in the sample database tables that are not used in this example, the reason is that this is part of an actual implementation that I use in one of my installations. I have removed all non-relevant code from the example. There are some additions that I have left in the example just for fun.

There is a very simple interface for recording which groups a particular user is part of into the user table. The groups are named and stored in a text array and then stuffed into a BLOB in the user table. It is very simple and fit perfectly for how it is used in my situation. I have left it there just for your interest.

Personally, I love quotes. All my systems have a "quote of the day" in the login dialogs. I have left this in also for your amusement.

I have used the article titled "Prescience in Databases" by Tom Kemp to further enhance the login dialog by having a type-ahead user name. Works really well and I am indebted to Tom for his article. I have simplified it a little since I didn't need it as generically as it was presented. Hopefully this article will do the same for a reader also.

The beauty of this login dialog is that it is quite flexible. At its core, the only fields that are needed are the [Users]UserID and the [Users]PasswordDigest fields. All the rest are just for additional information that might be used elsewhere in the system. You could even throw out the user table and use 4D's built-in password system to store the UserID (as a string of course) and the password digest as the 4D password. Leave the designer password blank and you don't have to use ACI's password dialog but your own.

## Drawbacks

The largest drawback that I have seen to date is that when using a multi-user system, 4D Server thinks that everyone is the designer. I have not figured out yet how to report which user is actually using a locked record. That is for the future.

## Conclusion

This login dialog has solved a specific problem for me. Mostly around keeping the users out of the structure and in the data file. There are a number of variations that I have implemented, all are relevant to my clients needs. There is an incredibly amount of flexibility in using your own password system as compared to ACI's. If you need more than a user ID and a password, then this might be just for you.

Please feel free to send me an email if you have any questions or if you would like me to send you a copy of the sample database that I put together.

## About the author

Dani Beaubien has been using 4D since 1994. He currently devotes most of his 4D expertise to developing really cool special education solutions for EIS, located in Carrboro, NC, although he loves living in Edmonton, Alberta, Canada. He can be contacted at *dbeaubien@infoHandler.com*.