Passwords are better with salt - hashing, salting and key stretching in kdb+

John Ludlow

July 3, 2014

1 Introduction

Password security is often a weak link in hardening systems against intrusion, as can be seen by the many reports of high profile breaches, e.g. linkedin
http://www.computerworld.com/s/article/9227869/Hackers_crack_more_than_60_of_breached_LinkedIn_passwords
and sony
http://www.darkreading.com/attacks-and-breaches/sony-hacked-again-1-million-passwords-exposed/d/d-id/1098113?.
With 32-bit kdb+ now free for commercial or educational use
http://kx.com/software-download.php
it is timely to look at best practices in password security. In this article the main focus will be on the storage and verification of user passwords. It is a large subject area beyond the confines of this article to fully cover, so links for the interested reader will be provided throughout.
A good overview of the history and future of password security can found at
http://www.openwall.com/presentations/Passwords12-The-Future-Of-Hashing/
Some useful background reading on password security is covered in:
https://crackstation.net/hashing-security.htm
http://security.blogoverflow.com/2013/09/about-secure-password-hashing/
http://security.stackexchange.com/questions/211/how-to-securely-hash-passwords/31846#31846

In this article, password hashing via MD5, SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512 will be looked at. The need for salting passwords via a cryptographically secure random number generator is then introduced, leadng to a discussion of key stretching via PBKDF2.

For other aspects necessary to consider when building an access control layer in kdb+, such as user classes and how to filter incoming queries, see

http://www.firstderivatives.com/lecture_series_pp.asp?downloadflyer=q_for_Gods_July_2013.

Kdb+ 3.1 2014.05.21 (32 bit) was used for all examples described herein.

2 Linking kdb+ to a cryptographic library

There are many freely available cryptography libraries that could be linked to kdb+ in a similar way to the examples to be discussed in this article. Some of the more popular libraries are listed at

http://en.wikibooks.org/wiki/Cryptography/Open_Source_Cryptography

Here, openssl will be used. To download and install openssl, see:

http://www.openssl.org/

http://http//www.geeksww.com/tutorials/libraries/openssl/installation/installing_openssl_on_ubuntu_linux.php

It is also noted that a kdb+ wrapper around libcurl which is commonly used for secure file transfers (SSL/TLS) using openssl has been released

http://curl.haxx.se/libcurl/

http://kx.com/q/c/c/curl.c

In this article a small set of wrapper functions will be used to interface kdb+ and certain openssl functions. The c code for this, qcrypt.c, can be compiled as follows (alter linking part of the compile line to suit particular openssl version, library location)

This produces a shared object qcrypt.so comprising of three functions qrand, hash and pbkdf2 that can then be loaded into kdb+

```
KDB+ 3.1 2014.05.21 Copyright (C) 1993-2014 Kx Systems
l32/ 1()core 502MB johnl johnl 127.0.1.1 NONEXPIRE

Welcome to kdb+ 32bit edition
For support please see http://groups.google.com/d/forum/personal-kdbplus
Tutorials can be found at http://code.kx.com/wiki/Tutorials
To exit, type \\
To remove this startup msg, edit q.q
q)qrand:`qcrypt 2: (`qrand;1)
q)hash:`qcrypt 2: (`hash;2)
q)pbkdf2:`qcrypt 2: (`pbkdf2;4)
q)
q)
q)
```

For details on interfacing kdb+ with C, see

http://code.kx.com/wiki/Cookbook/ExtendingWithC

http://code.kx.com/wiki/Cookbook/InterfacingWithC

3 Hashes

A cryptographic hash is a one-way function that scrambles an input string. Kdb+ does have a built in hash function, namely md5 (Message-Digest algorithm 5), see

http://code.kx.com/wiki/Reference/md5

http://en.wikipedia.org/wiki/MD5

http://tools.ietf.org/html/rfc1321

However md5 is no longer recommended for serious cryptographic protection due to weaknesses in the algorithm

http://www.win.tue.nl/hashclash/rogue-ca/

http://blogs.technet.com/b/srd/archive/2012/06/06/more-information-about-the-digital-certificates-used-to-sign-the-flame-malware.aspx

http://en.wikipedia.org/wiki/Flame_(malware)

As a first test of the kdb+ to openssl interface, the md5 function built into kdb+ will be compared to the corresponding md5 function in openssl

https://www.openssl.org/docs/crypto/md5.html

```
KDB+ 3.1 2014.05.21 Copyright (C) 1993-2014 Kx Systems
l32/ 1()core 502MB johnl johnl 127.0.1.1 NONEXPIRE

Welcome to kdb+ 32bit edition
For support please see http://groups.google.com/d/forum/personal-kdbplus
Tutorials can be found at http://code.kx.com/wiki/Tutorials
To exit, type \\
To remove this startup msg, edit q.q
q)hash:`qcrypt 2: (`hash;2)
q)md5"thisisatest"
0xf830f69d23b8224b512a0dc2f5aec974
q)hash["thisisatest";"md5"]
0xf830f69d23b8224b512a0dc2f5aec974
q)
```

The first argument to the qcrypt 'hash' is the input string and the second argument is the hashing algorithm.

A stronger set of hash functions is the SHA group of algorithms - SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512. See

http://tools.ietf.org/html/rfc6234

http://en.wikipedia.org/wiki/Sha-1

http://en.wikipedia.org/wiki/SHA-2

For their implementation in openssl, see

https://www.openssl.org/docs/crypto/sha.html

The kdb+ results are shown below

```
KDB+ 3.1 2014.05.21 Copyright (C) 1993-2014 Kx Systems
l32/ 1()core 502MB johnl johnl 127.0.1.1 NONEXPIRE

Welcome to kdb+ 32bit edition
For support please see http://groups.google.com/d/forum/personal-kdbplus
Tutorials can be found at http://code.kx.com/wiki/Tutorials
To exit, type \\
To remove this startup msg, edit q.q
q)\c 2000 2000
q)hash:`qcrypt 2: (`hash;2)
q)hash["thisisatest";"sha1"]
0x42d4a62c53350993ea41069e9f2cfdefb0df097d
q)hash["thisisatest";"sha224"]
0x8143b8f30ad19141ed2aade0fc7eff74cd39d2b439ce5f33b665f688
q)hash["thisisatest";"sha256"]
0xa7c96262c21db9a06fd49e307d694fd95f624569f9b35bb3ffacd880440f9787
q)hash["thisisatest";"sha384"]
0x2cb5eb7d54da9594424bf4a375dd2e1b678668f74d8a88d8747f58ab73bfc0afb8ddda07a3f9a2
4315c16a306c848821
q)hash["thisisatest";"sha512"]
0xd44edf261feb71975ee9275259b2eab75920d312cb1481a024306002dc57bf680e0c3b5a00edb6
ffd15969369d8a714ccce1396937a57fd057ab312cb6c6d8b6
q)
```

To check the results, there are online calculators that can used, e.g.

http://www.miniwebtool.com/hash-and-checksum/

SHA1 Hash Generator

Enter the text:

thisisatest

Generate SHA1 Hash

SHA1 Hash of Input String

42d4a62c53350993ea41069e9f2cfdefb0df097d

SHA224 Hash Generator

Enter the text:

thisisatest

Generate SHA224 Hash

SHA224 Hash of Input String

8143b8f30ad19141ed2aade0fc7eff74cd39d2b439ce5f33b665f688

SHA256 Hash Generator

Enter the text:

thisisatest

Generate SHA256 Hash

SHA256 Hash of Input String

a7c96262c21db9a06fd49e307d694fd95f624569f9b35bb3ffacd880440f9787

SHA384 Hash Generator

Enter the text:

thisisatest

Generate SHA384 Hash

SHA384 Hash of Input String

2cb5eb7d54da9594424bf4a375dd2e1b678668f74d8a88d8747f58ab73bfc0afb8ddda07a3f9a24315c16a306c848821

SHA512 Hash Generator

Enter the text:

thisisatest

Generate SHA512 Hash

SHA512 Hash of Input String

d44edf261feb71975ee9275259b2eab75920d312cb1481a024306002dc57bf680e0c3b5a00edb6ffd15969369d8a714ccce1396937a57fd057ab312cb6c6d8b6

4 Salt

A hash of a password by itself is not sufficient to guarantee security. User passwords are often chosen insecurely and are amenable to dictionary attacks

http://en.wikipedia.org/wiki/Dictionary_attacks

which compare the hashes of common passwords against the hash of the user password. It is also possible to pre-compute hashes and then check them against stored passwords via rainbow tables

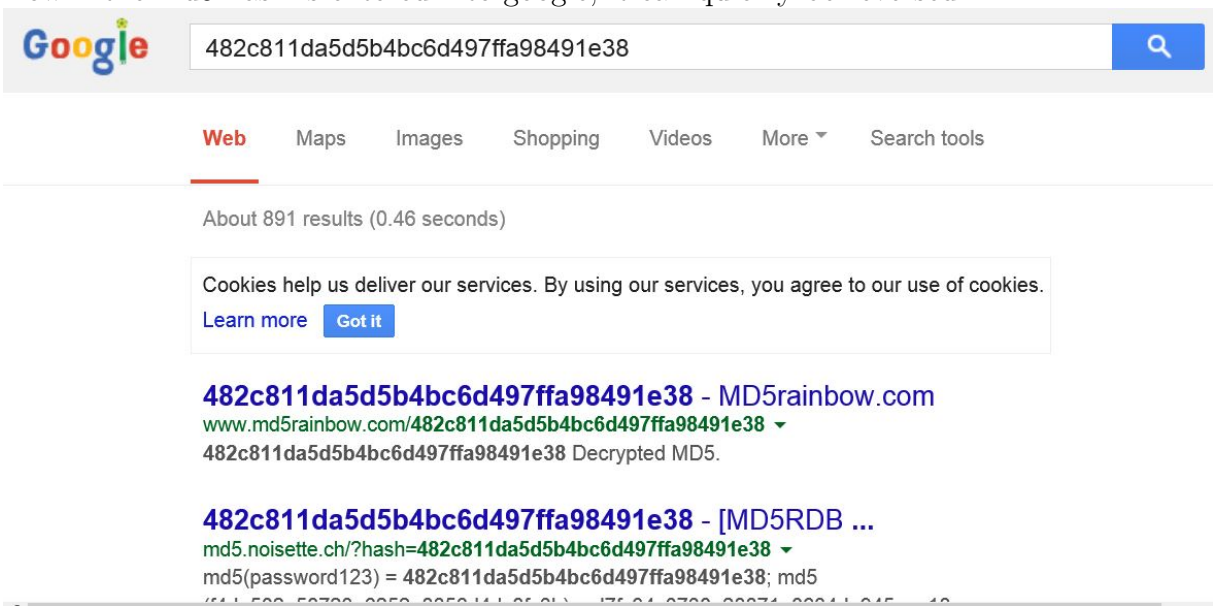
http://en.wikipedia.org/wiki/Rainbow_table

To illustrate this consider the md5 hash of the string "password123".

```
KDB+ 3.1 2014.06.03 Copyright (C) 1993-2014 Kx Systems
w32/ 2()core 4095MB johnl johnl-pc 192.168.56.1 NONEXPIRE

q)md5 "password123"
0x482c811da5d5b4bc6d497ffa98491e38
q)
```

Now if the md5 hash is entered into google, it can quickly be reversed



In order to make such attacks harder and more costly to the attacker, a random salt can be added to the password before it is hashed. As long as a unique salt is generated per password, this means that each password must be attacked individually

http://en.wikipedia.org/wiki/Salt_(cryptography)

It is important that the salt is produced using a cryptographically secure random number generator. Here the openssl function RAND_bytes will be used

https://www.openssl.org/docs/crypto/RAND_bytes.html

http://wiki.openssl.org/index.php/Random_Numbers

On linux, this uses /dev/urandom or /dev/random as sources of entropy to seed a pseudo-random number generator. The kdb+ function 'qrand' takes a single argument, the desired number of random output bytes

```
KDB+ 3.1 2014.05.21 Copyright (C) 1993-2014 Kx Systems
l32/ 1()core 502MB johnl johnl 127.0.1.1 NONEXPIRE

Welcome to kdb+ 32bit edition
For support please see http://groups.google.com/d/forum/personal-kdbplus
Tutorials can be found at http://code.kx.com/wiki/Tutorials
To exit, type \\
To remove this startup msg, edit q.q
q)qrand:`qcrypt 2: (`qrand;1)
q)qrand(10)
0x4da2ef2116061d6af53a
q)qrand(20)
0x731807c68be02bb76d693c1611e4fb37209ac3d8
q)qrand(20)
0x4395647314f93966ed4247ca5acca98be622a938
q)qrand(20)
0x9b1ff7ce6610a99243f7d32d7ffd579951ed130e
q)count qrand(20)
20
q)
```

5 Key Stretching

As has been seen, simply hashing an input password is not enough to stop an attacker. Adding a salt to the password before hashing strengthens security. By then iterating the hash of the salted password in a process known as 'key stretching', attacks can be made much more computationally expensive. There are various key stretching algorithms such as pbkdf2, bcrypt and scrypt.

http://en.wikipedia.org/wiki/PBKDF2

http://en.wikipedia.org/wiki/Bcrypt

http://en.wikipedia.org/wiki/Scrypt

Here the pbkdf2 (Password-Based Key Derivation Function 2) algorithm will be used, see

http://www.ietf.org/rfc/rfc2898.txt

http://www.openssl.org/docs/crypto/PKCS5_PBKDF2_HMAC.html

http://stackoverflow.com/questions/9771212/how-to-use-pkcs5-pbkdf2-hmac-sha1

The kdb+ function 'pbkdf2' takes 4 arguments - the password, salt, number of iterations and the length of the derived output key. Example usage is shown below,

```
KDB+ 3.1 2014.05.21 Copyright (C) 1993-2014 Kx Systems
l32/ 1()core 502MB johnl johnl 127.0.1.1 NONEXPIRE

Welcome to kdb+ 32bit edition
For support please see http://groups.google.com/d/forum/personal-kdbplus
Tutorials can be found at http://code.kx.com/wiki/Tutorials
To exit, type \\
To remove this startup msg, edit q.q
q)pbkdf2:`qcrypt 2: (`pbkdf2;4)
q)\c 2000 2000
q)pbkdf2["thisisatest";`byte$"salt";100;20]
0x04272da9bcf406ab081bd2f1e9ae114834589bce
q)pbkdf2["password";`byte$"salt";100;50]
0x8595d7aea0e7c952a35af9a838cc6b393449307cfcc7bd340e7e32ee901156509cd8f76e53cf8c
c09d851fd2da1388ce4518
q)
```

This can be compared to an online pbkdf2 calculator such as

http://anandam.name/pbkdf2/

A higher number of iterations of the pbkdf2 algorithm provides more security. For example in tests it takes about a second to run the pbkdf2 function with a 512 byte salt, 25000 iterations and a 512 byte output key. However note the time versus security trade off here in that more security comes at the cost of more time spent in validating user connection requests.

```
KDB+ 3.1 2014.05.21 Copyright (C) 1993-2014 Kx Systems
l32/ 1()core 502MB johnl johnl 127.0.1.1 NONEXPIRE

Welcome to kdb+ 32bit edition
For support please see http://groups.google.com/d/forum/personal-kdbplus
Tutorials can be found at http://code.kx.com/wiki/Tutorials
To exit, type \\
To remove this startup msg, edit q.q
q)pbkdf2:`qcrypt 2: (`pbkdf2;4)
q)qrand:`qcrypt 2: (`qrand;1)
q)salt:qrand[512]
q)\t pbkdf2["password";salt;25000;512]
1066
q)
```

6 Putting it all together

Now the various functions in qcrypt can be combined together to form the first stage of a simple access control layer. As mentioned in the introduction, a complete acl layer would also need to account for user classes and filtering of user function calls.

The q script access.q provides a number of functions to add, delete and update user passwords and to verify incoming access requests. The default setting for the hashing algorithm (.acl.HASHFN) is pbkdf2 with a salt length (.acl.SALTLEN) of 512 bytes, 25000 iterations (.acl.ITERATIONS) and a derived key length (.acl.DKLEN) of 512. These default settings can be overridden by changing the 'saltlen', 'iterations', 'dklen' and 'hashfn' parameters in settings.csv. The 'hashfn' parameter can take the values: md5, sha1, sha224, sha256, sha384, sha512 and pbkdf2.

```
/ define default configuration settings
.acl.SALTLEN:512; / salt length - ensure it is as long as the maximum hash length
.acl.ITERATIONS:10000; / number of iterations for pbkdf2 algorithm
.acl.DKLEN:512;      / derived key length for pbkdf2 algorithm
.acl.HASHFN:`pbkdf2; / hash algorithm to be used
.acl.users:([u:`$()] p:();s:()); / users table schema
system["c 2000 2000"];
```

```
johnl@johnl:~/kdb$ cat settings.csv
saltlen,512
iterations,25000
dklen,512
hashfn,pbkdf2
johnl@johnl:~/kdb$
```

Usernames and passwords are stored in a file users.csv. Note that if you change settings.csv you will have to re-generate all saved password hashes, so no function is provided for this in order to reduce the risk of unplanned changes to the algorithm settings.

The message handler .z.pw is used to verify incoming user passwords against the stored hash.

Some examples are shown below to illustrate usage. The salt and pbkdf2 key lengths are reduced here to 10 with the number of iterations set to 100 iterations so as to make usage easier to demonstrate:

- Users can be added using the function .acl.addUser that takes two input arguments, a string for the username and a string for the password. The function then generates a random salt, encrypts a concatenation of the salt and password using the function .acl.enCrypt and then upserts it to the keyed table .acl.users that has columns for the user, password hash and salt. The users table is then saved down to a csv file,

users.csv. The users.csv and settings.csv files are read when the access.q script is loaded

```
johnl@johnl:~$ q kdb/access.q
KDB+ 3.1 2014.05.21 Copyright (C) 1993-2014 Kx Systems
l32/ 1()core 502MB johnl johnl 127.0.1.1 NONEXPIRE

Welcome to kdb+ 32bit edition
For support please see http://groups.google.com/d/forum/personal-kdbplus
Tutorials can be found at http://code.kx.com/wiki/Tutorials
To exit, type \\
To remove this startup msg, edit q.q
q)\f .acl
`addUser`delUser`enCrypt`hash`pbkdf2`qrand`reload`toString`toSymbol`userChk
q)\c 2000 2000
q).acl.users
u| p s
-| ---
q).acl.addUser["bob";"bobspassword"]
q).acl.users
u  | p                        s
---| ----------------------------------------
bob| 0xc8312d1062eb5f26c0cb 0x3f9e342db8474db2d78c
q)\more kdb/users.csv
"u,p,s"
"bob,c8312d1062eb5f26c0cb,3f9e342db8474db2d78c"
q)
```

```
q).acl.addUser
{[user;pass]
    user:.acl.toSymbol[user];
    salt:.acl.qrand[.acl.SALTLEN];
    `.acl.users upsert (user;.acl.enCrypt[salt;pass];salt);
    .acl.usersFile 0: csv 0: 0!update raze each string[p],raze each string[s] from .ac
l.users;
 }
q).acl.enCrypt
{[salt;pass]
    if[.acl.HASHFN in .acl.hashes;:.acl.hash[;string .acl.HASHFN] raze .acl.toString s
alt,pass];
    if[.acl.HASHFN~`pbkdf2;:.acl.pbkdf2[pass;salt;.acl.ITERATIONS;.acl.DKLEN]];
 }
q)
```

- Similarly if .acl.addUser is ran where the username matches an existing username, the entry for that user is updated with a new password. The function .acl.delUser takes a single input symbol for the username to be deleted.

```
To remove this startup msg, edit q.q
q).acl.users
u  | p                    s
---| ----------------------------------------
bob| 0xc8312d1062eb5f26c0cb 0x3f9e342db8474db2d78c
q).acl.addUser["alice";"alicespassword"]
q).acl.users
u    | p                    s
-----| ----------------------------------------
bob  | 0xc8312d1062eb5f26c0cb 0x3f9e342db8474db2d78c
alice| 0x8eb13c43bc60b3b4c4d1 0xd929d1c4d57b6dc8a79c
q).acl.delUser[`bob]
q).acl.users
u    | p                    s
-----| ----------------------------------------
alice| 0x8eb13c43bc60b3b4c4d1 0xd929d1c4d57b6dc8a79c
q)\more kdb/users.csv
"u,p,s"
"alice,8eb13c43bc60b3b4c4d1,d929d1c4d57b6dc8a79c"
q).acl.addUser["alice";"alicesnewpassword"]
q).acl.users
u    | p                    s
-----| ----------------------------------------
alice| 0x2d17d8fbb57ae9f30c3a 0x5014b20becae8169588e
q)\more kdb/users.csv
"u,p,s"
"alice,2d17d8fbb57ae9f30c3a,5014b20becae8169588e"
q)
```

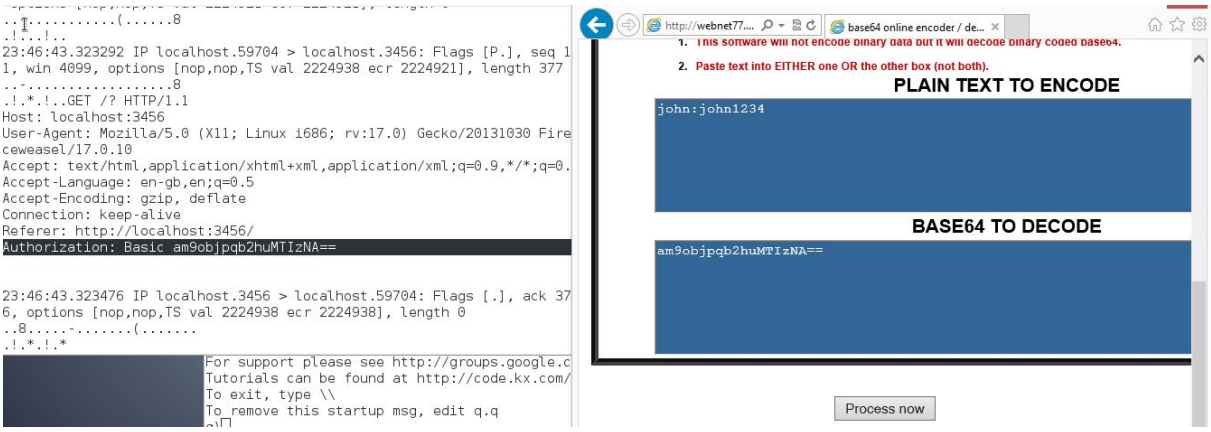
```
q).acl.delUser
{[user]
    delete from `.acl.users where u=user;
    .acl.usersFile 0: csv 0: 0!update raze each string[p],raze each string[s] from .ac
l.users;
 }
q)
```

- Now a test connection can be made from another q session. Note that as well as .z.pw, other message handlers such as .z.po, .z.pc, .z.pg, .z.ps would also be overloaded in a full system.


johnl@johnl:~$
johnl@johnl:~$
johnl@johnl:~$
johnl@johnl:~$
johnl@johnl:~$
johnl@johnl:~$
johnl@johnl:~$
johnl@johnl:~$
johnl@johnl:~$
johnl@johnl:~$
johnl@johnl:~$
johnl@johnl:~$
johnl@johnl:~$ q
KDB+ 3.1 2014.05.21 Copyright (C) 1993-2014 Kx Systems
l32/ 1()core 502MB johnl johnl 127.0.1.1 NONEXPIRE
Welcome to kdb+ 32bit edition
For support please see http://groups.google.com/d/forum/personal-kdbplus
Tutorials can be found at http://code.kx.com/wiki/Tutorials
To exit, type \\
To remove this startup msg, edit q.q
q)h:hopen `:localhost:1234:alice:alicesnewpassword
q)h"1+1"
2
q)h"test:4"
q)
ystems
l32/ 1()core 502MB johnl johnl 127.0.1.1 NONEXPIRE
Welcome to kdb+ 32bit edition
For support please see http://groups.google.com/d/forum/personal-kdbplus
Tutorials can be found at http://code.kx.com/wiki/Tutorials
To exit, type \\
To remove this startup msg, edit q.q
q)\p
1234i
q).z.pw
{[user;pass]
$[.acl.userChk[user;pass]~.acl.users[.acl.toSymbol[user]][`p];1b;0b]
}
q).acl.users
u	p s
alice| 0x2d17d8fbb57ae9f30c3a 0x5014b20becae8169588e
q)test
4
q)

7 Network Security

There is a loophole in the password encyption techniques discussed in the preceding sections, namely there is an implicit assumption that all communications are taking place over a secure network. If the network connection is untrusted, then kdb+ passwords can easily be retrieved. This can be done be using packet sniffers such as tcpdump to listen to network traffic. This is illustrated below for a password sent to a kdb+ server from a web browser where the password is sent over the network as unencrypted base64 encoded bytes.



There a couple of solutions to this. SSL/TLS can be used to encrypt the network con-

nection

https://www.openssl.org/related/ssl.html

An alternative is the use of Kerberos

http://web.mit.edu/kerberos/

allowing a single sign solution. Note that combining Kerberos with LDAP is often a good combination with Kerberos used to authenticate clients and LDAP used for authorization, see

https://wiki.debian.org/LDAP/Kerberos

Ensuring network security for communicating kdb+ processes will be the subject of a future guide...