MWA Software

IBX for Lazarus and Firebird Security

Issue 1.0,
7 February 2018

McCallum Whyman Associates Ltd

EMail: info@ mccallumwhyman.com, http://www.mccallumwhyman.com

Registered in England Registration No. 2624328

COPYRIGHT

Disclaimer

Although our best efforts have been made to ensure that the information contained within is up-to-date and accurate, no warranty whatsover is offered as to its correctness and readers are responsible for ensuring through testing or any other appropriate procedures that the information provided is correct and appropriate for the purpose for which it is used.

CONTENTS **Page**

1

Introduction

This document is a supplement to the IBX For Lazarus User Guide and explores how *IBX for Lazarus* can be used to create applications that manage Firebird users and their access rights to Firebird Databases.

Previously to Firebird 3, there was a single (global) security database (containing user credentials) per server, and the Services API was used to perform User Management. Many access rights were implicit (e.g. creating tables or even whole databases) and the granting and revoking of access rights was performed using DDL statements.

Firebird 3 has improved upon this by:

- Permitting User Management to be performed using a combination of virtual tables and DDL Statements.
- Supporting alternative security databases on a per database basis.
- Allowing all access rights to be explicitly managed through DDL statements.

For legacy support, the Services API is still available for User Management. It is limited to the global security database.

Through the IBServices unit, IBX has made available access to the Firebird Services API and hence could support User Management applications, with the supporting DDL Statements being executed using TIBSQL. This functionality continues to be available.

In IBX 2.1, the TIBUpdate component was introduced. This is intended to support dataset update using DDL statements. That is datasets generated from Firebird virtual tables and presented to the user using TIBQuery. Together they enable Firebird 3 style User Management through virtual tables and supporting DDL statements in a straightforward manner.

The *IBX for Lazarus* source code provides examples for both legacy and Firebird 3 User Management. This guide supports these examples and attempts to:

- explain the Firebird Security Model
- How IBX is used to support legacy User Management through the Services API.
- How IBX is used to support Firebird 3 User Management and the assignment of extended Access Rights

1.1 References

1. IBX for Lazarus User Guide -MWA Software – Issue 1.4 https://mwasoftware.co.uk/downloads/send/5-ibx-current/147-ibx4lazarusguide
2. Firebird 3.0.2 Release Notes https://www.firebirdsql.org/file/documentation/release_notes/html/en/3_0/rlsnotes30.html
3. Firebird 2.0.7 Release Notes https://www.firebirdsql.org/file/documentation/release_notes/html/rlsnotes207.html
4. Firebird 2.1.7 Release Notes https://www.firebirdsql.org/file/documentation/release_notes/html/rlsnotes217.html
5. Firebird 2.5.8 Release Notes https://www.firebirdsql.org/file/documentation/release_notes/html/en/2_5/rlsnotes25.html
6. Firebird 2.5 Language Reference Update https://www.firebirdsql.org/refdocs/langrefupd25.html
7. Firebird 1.5.6 Release Notes https://www.firebirdsql.org/file/documentation/release_notes/html/rlsnotes15.html
8. Firebird 2.5 Language Reference https://www.firebirdsql.org/file/documentation/reference_manuals/fblangref25-en/html/fblangref25.html
9. US Department Of Defense Trusted Computer System Evaluation Criteria- DoD 5200.28-STD – December 1985
10. RFC 2945 The SRP Authentication and Key Exchange System, September 2000.
11. Wu, T., "SRP-6: Improvements and Refinements to the Secure Remote Password Protocol", Submission to IEEE P1363.2 working group, October 2002, http://srp.stanford.edu/srp6.ps. See also: http://srp.stanford.edu/design.html
12. The Windows Security Support Provider Interface (SSPI) https://msdn.microsoft.com/en-us/library/windows/desktop/aa380493(v=vs.85).aspx
13. RFC 5054 Using the Secure Remote Password (SRP) Protocol for TLS Authentication, November 2007.
14. ISO/IEC 27001 Information technology — Security techniques — Information security management systems — Requirements

15. How to use SRP in Openssl – https://matthewarcus.wordpress.com/2014/05/10/srp-in-openssl/

2

Firebird Security

Before discussing how IBX for Lazarus can be used to manage Firebird users, it is useful to have an appreciation of Firebird Security. The aim of this section is to present an understanding of how Firebird Databases are Secured. This sections collects together information from many sources including Release Notes for different versions of Firebird.

The reader's attention is also drawn to the disclaimer at the start of this document. Where information security is concerned, no person or organisation should rely upon the statements made in this or any other document. When security mechanisms or other assumptions are being relied upon to protect sensitive data, they should always be subject to a program of verification and testing in order to verify that those mechanism perform their intended task and that any assumptions are valid and appropriate. Those seriously interested in information security may also wish to read ISO 27005 **[14]**. The reader should also note that this review is not necessarily complete in that it is not a general review of server and data communications security and that there are other subjects (e.g. intrusion detection) that are outside of this scope of this review.

Note: throughout this section, endnotes are used to clarify various statements and to provide source references, in addition to direct references to documents. Direct references are in bold and enclosed in square brackets, while superscripts are used for endnotes.

2.1 Overview

In order to protect sensitive user data Firebird implements a form of Discretionary Access Control (DAC) which itself has been “defined by the Trusted Computer System Evaluation Criteria **[9]** as:

> A means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject (unless restrained by mandatory access control).

These criteria require both “an enforcement mechanism (e.g., self/group/public controls, access control lists)” and require users both to identify themselves and to “use a protected mechanism

(e.g., passwords) to authenticate the user's identity. The TCB shall protect authentication data so that it cannot be accessed by any unauthorized user."

Protecting User Data

Both the need to provide an "enforcement mechanism" and to "protect authentication" data demand at least either:

a) that Firebird Servers rely upon physical access controls and the Host File System to protect local access to databases, or

b) Firebird databases and authentication data is encrypted such that they are only available to authorised users.

That latter approach is appropriate for unsecured servers, such as cloud servers while, the former, may be appropriate when the database server is housed in a secure environment managed by the data owner.

Note: In either case, there is still a vulnerability to data loss whether through malicious action or system failure that needs to be countered through backups and other copies of the database being held at physically separate locations.

Access Modes

Firebird Servers provide two modes of user access to data:

a) Embedded: where the server runs as part of the user process, and

b) Remote: where some data communications mechanism is used to support data exchange between the server and a remote client. TCP/IP is the most most common communications mechanism used; on Windows a local IPC based mechanism (xnet) is also in use.

Embedded mode has been supported by Firebird since release 1.5 [**7]**, although a similar capability could be found in some earlier versions of InterBase. In embedded mode, the server gives the user full access to any database that it can open and hence file system controls are the only mechanism used to protect access to user data in embedded mode. It is suitable for Personal Databases and is also necessary for some general administration tasks. On a general purpose server, it can readily provide a means to circumvent data access controls if file permissions are not set appropriately.

Remote Access

Remote Access is the typically means by which most users access Firebird Databases. Remote Access is not only useful in permitting access from remote systems, it also allows the Database Server itself to control access to individual objects within the database, in the line with the DAC; something that is just not enforcible in embedded mode. From Firebird 3 onwards, Firebird also supports "over the write encryption" in support of remote access to user data where the communications path is vulnerable to attack[1].

The Firebird Server provides remote access while running in SuperServer, SuperClassic or Classic mode. The release notes **[2]** describe each of these modes. However, for reasons due to conflicting locking strategies, the SuperServer mode should be avoided if concurrent use of embedded and remote access is deployed.

The Firebird Server also provides two independent APIs for Remote Access. These are:

- The Services API: used to perform administrative tasks including User Management, Backup and Restore, Statistics collection, etc.

- The Database API: used to support connections to databases using SQL.

In Firebird Remote Access Mode, Discretionary Access Rights are enforced such that:

- The objects to which access is controlled are items such as a Database Table and its data, and Stored Procedures.

- The subjects include Firebird “users” as well as “objects” taking on the role of a subject (e.g. Stored Procedures and Triggers).

- The groups (or access control lists) correspond to SQL Roles. There is also special group called 'PUBLIC' that all users implicitly belong to.

- The access rights themselves include the rights to access and change database data and/or metadata[2].

- All Firebird objects have an “owner” - a Firebird user – and which is able to grant subject access rights on the object. However, the permission to grant rights can also be passed on to other users.

Firebird Users and Roles

In Remote Access Mode, each Firebird user is identified through a unique login “UserName” (limited to 31 characters and constrained to the same character set as an SQL Identifier) and who must be authenticated (logged on) when a connection is established to the database or to the Services API. Firebird also includes to notion of a “SuperUser”. That is a user that has full access rights to all objects. Firebird calls this user “SYSDBA”.

In line with SQL standards, a database designer can create named “user groups” (known as “roles” in SQL) and assign access rights to roles rather than to individual users. Users can be “granted access to a role” which permits them to logon under that role (by including a requested role name in their user credentials) and then assume all access rights granted to the role.

In general, this should be preferred to granting rights to individual users as it can greatly simplify database administration and reduces the risk of individual users keeping access rights that are inappropriate for their current use of the database.

Firebird 2.5 introduces a built-in role “RDB$ADMIN” **[5]**. A user granted this role at the database level has SuperUser rights to all objects in that database. A user may also be granted this role at the server level and hence have the privilege to manage user accounts.[3]

Access Rights

Access Rights on objects are granted to subjects/groups using DDL Grant and Revoke statements on a per database basis. Access rights can only be granted when a user is (a) logged on to a database and (b) is the object owner or has been given permission by the object owner to grant the right, or is the SYSDBA.

For example:

```
Grant Select on MyTable to Alice;
```

grants the “Select” access right to a user called “Alice” to a table called “MyTable”. The select access right allows Alice read access to the data stored in MyTable.

Note: DDL statements cannot have placeholder parameters and while they can be executed by TIBSQL, the SQL statement has to be formatted and assigned to the component each time it is used.

Prior to Firebird 3, access rights to metadata were implicit and permissive[4]. Any Firebird user could create an object and become the object's owner. In Firebird 3, the DAC has been extended to the metadata, with create/alter/drop access rights being available on metadata objects. There is also an explicit “create database” permission that a user must be given in order to be allowed to create a new database.

2.2 Reliance on File System Security

As discussed above, when Firebird Databases are located on secure servers under the control of the data owner, there is no strong need to encrypt the database. Indeed, this was not even possible prior to Firebird 3. However, even with physically secure servers, there is still a need to rely on file system security in order to protect user data.

Each Firebird Database comprises one or more files protected by the host file system. If the file system is compromised or its settings are weak enough to allow compromise then it is possible for an unauthorised user to:

- Copy a Database and read its data on another system with a Firebird Server.
- Modify or delete a database *in situ*.

Assuming that the File System controls are themselves good, provide Discretionary Access Controls (for a multi-user environment), and are resistant to attack, these potential vulnerabilities can be mitigated by the following:

1. Creating an Operating System User Account dedicated to the Firebird Server (e.g. under Linux, this is typically the “no-login” account “firebird”), and that this account also “owns” all Firebird configuration files and databases.
2. Running the Firebird Server under the “firebird” account.
3. Ensuring that the Firebird configuration files are writeable only by the “firebird” user (or an authorised System Administrator).
4. All Firebird databases and authentication data are readable and writeable only by the “firebird” user.
5. The File System settings for all directories in the file system path to each Firebird configuration file and database must not permit the creation, modification or deletion of files or sub-directories by unauthorised users.[5]

The Firebird configuration file “firebird.conf” has a “DatabaseAccess” setting that can limit where Firebird databases may be located. This should be used to limit these locations such that the Database Administrator has ensure that they are compliant with item 5 above. This avoids the risk of users creating databases in vulnerable locations.

Indeed, some may want to go further and set “DatabaseAccess = None” in the Firebird Configuration File. This means that the only databases that may be created or accessed are those which have aliases and explicit path names listed in the “databases.conf” file[6]. This can also be used to hide database file system locations – as long as the “databases.conf” file is not readable by unauthorised users.

2.3 Database Encryption

A Database Encryption capability was introduced in Firebird 3 and is potentially useful in situations where a database cannot be fully protected from unauthorised access, such as on a “cloud server”, or just as an extra level of security where sensitive data is concerned.

The “capability” is to support a “plugin” that may be used to encrypt or decrypt a database. There is no default database encryption plugin, although third party plugins are becoming available.

A key issue with database encryption is key management and specifically how the encryption key is communicated to the plugin. It is clearly undesirable to keep the key in a file on the server, as that would defeat the objective of protecting access to user and authentication data on an unsecured server. Instead, it must be communicated when the user logs on.

In turn, this demands that the client is able to authenticate the server and that database encryption key is protected when it is communicated to the server (e.g. by over the wire encryption).

Note: if the client has no means of authenticating the server, then an attacker may be able to masquerade as a genuine server and solicit the database encryption key from an unsuspecting client.

A “KeyHolder” plugin capability also exists to support database encryption key exchange from client to server.

2.4 Firebird Users and User Authentication

A key requirement for Remote Access to Firebird Databases is that there must exist a means to identify and authenticate remote users.

Prior to Firebird 3, there was a single authentication mechanism based on a simple password exchange. This is now known as “legacy authentication”. Firebird 3 allows for multiple authentication plugins each providing a different means of authenticating users directly or indirectly using an external authentication server. A Database Administrator can select the plugin(s) that are permitted to authenticate users via the “AuthServer” setting in the “firebird.conf” file. Those which are client may use to request authentication by are likewise enumerated in the “AuthClient” setting.

When user authentication is performed by the plugin itself, there needs to be a means of securely storing user authentication data on the server. A common means of storing such data is provided by the Firebird Security Database.

2.4.1 The Firebird Security Database

Prior to Firebird 3, each Firebird Server maintained a single local security database – now known as the “default security database”. This security database is a normal Firebird Database and typically contained a single table called RDB$USERS (or even just “USERS” in earlier versions). The Firebird Server (authentication plugin) reads the security database whenever it needs to authenticate a user login. Maintaining the integrity and security of the data held in the security database is thus critical to managing a secure Firebird Server.[7]

Prior to Firebird 2.0, the security database was directly maintained by the *gsec* utility and could be accessible to remote users. It was thus potentially available for read access by various users and hence exposed user credentials which, in turn, became vulnerable to a brute force attack on the password hashes stored in the database[8]. Firebird 2.0 changed *gsec* to use the Firebird Services API and hence permitted the security database to be hidden from user view, accessible only to the Firebird Server.[9]

In Firebird 3:

- the plugin architecture allows for multiple user authentication mechanisms and each can save its authentication data in its own tables in a Firebird Security Database.
- alternative security databases can also be defined on a per database level. That is the original Firebird Security Database is the default security database unless an alternative is defined for a given database.
- It is also possible for a user authentication plugin to save its authentication data in a user database[10]. This may well be desirable when the database is encrypted. The encryption mechanism can then be used to protect both the user data and the user authentication data.[11]

Firebird 3 has also restored the ability of users to establish a direct embedded connection to a security database (or even a remote connection if the server configuration is permissive enough). This is necessary to create alternative security databases. This restored capability needs to be managed with care if Firebird's previous vulnerabilities are not to be re-introduced. However, once a security database has been created (see 2.4.1.4), it can be hidden from view and only needs to be accessible to the Firebird Server.

2.4.1.1 Security Database Configuration

Prior to Firebird 3, the location of the security database was compiled into the server. In Firebird 3, the location is configurable.

The default configuration for the security database in Firebird 3 is split between “firebird.conf” and “databases.conf”:

- In “firebird.conf”, the “SecurityDatabase” setting is configured with the path to the “default security database”.
- In “databases.conf”. The alias “security.db” is assigned to the database and its local configuration settings include “RemoteAccess = false”. This setting prevents remote access. File system settings should limit access to authorised users only, and only a System Administrator should have embedded access.

Note: The “RemoteAccess” setting is generally useful as a means of limiting database access, for any database, to embedded mode only. It should always be set to “false” for a security database other than when it is embedded in a user database.

2.4.1.2 Creating a Firebird Security Database

Prior to Firebird 3, there was only a single Firebird Security Database and this was normally provided as part of the initial set of files when the server was installed. The location differed between platforms and OS distributions but the installed file was the same and included the RSB$USERS table with a single initial (super) user “SYSDBA” and a default password

“masterkey”. It was up to the Database Administrator (DBA) to change the SYSDBA password to something less well known and the create other users.

2.4.1.3 The Firebird 3 Default Security Database

With Firebird 3, such a simple strategy is no longer possible given that many different plugins could be available for user authentication. Different installers may take different strategies. Some may continue to provide a prepared version of the default security database with the SYSDBA user and a default password and set up for the default authentication plugin.

However, some may do no more than install an empty security database and leave it to the DBA to select the user authentication mechanism and create the SYSDBA user with an appropriate password. In many ways, this is better than installing the security database with a very well known password, and which includes the risk of being left in place if the DBA installs an alternative preferred user authentication plugin without fully disabling the default.

The “CREATE USER” DDL statement (introduced in Firebird 3) also makes it straightforward for the DBA to create the SYSDBA user even when the security database is empty. The process is described in the Firebird 3 release notes **[2]**:

> Initialization is performed in embedded mode using the *isql* utility. For an embedded connection, an authentication password is not required and will be ignored if you provide one. An embedded connection will work fine with no login credentials and “log you in” using your host credentials if you omit a user name. However, even though the user name is not subject to authentication, creating or modifying anything in the existing security database requires that the user be SYSDBA; otherwise, *isql* will throw a privilege error for the CREATE USER request.
>
> The SQL user management commands will work with any open database. Because the sample database `employee.fdb` is present in your installation and already aliased in `databases.conf`, it is convenient to use it for the user management task.
>
> 1. **Stop the Firebird server.** Firebird 3 caches connections to the security database aggressively. The presence of server connections may prevent *isql* from establishing an embedded connection.
>
> 2. In a suitable shell, start an *isql* interactive session, opening the employee database via its alias:
>
>    ```
>    > isql -user sysdba employee
>    ```
>
> 3. Create the SYSDBA user:
>
>    ```
>    SQL> create user SYSDBA password 'SomethingCryptic';
>    SQL> commit;
>    SQL> quit;
>    ```
>
> 4. To complete the initialization, start the Firebird server again. Now you will be able to perform a network login to databases, including the security database, using the password you assigned to SYSDBA.

If the SYSDBA already exists then “ALTER USER SYSDBA...” can be used instead to change the SYSDBA password. From Firebird 3.0.1 onwards the statement `CREATE OR ALTER USER SYSDBA PASSWORD <password>` can be used to initialize an empty security database. **[2]**

After it has been created, File System permissions must be used to protect the security database from unwanted attention (hopefully the installer will have done this for you):

- On a POSIX system, the Firebird Server usually operates under the user “firebird” and the security database is read/write to the “firebird” user (and possibly group) only with no other users permitted read or write access. In the above, *isql* needs to be run as user *root* in order to access the security database. After initialising the SYSDBA user, ensure that security database is owned (or at least read/write) by user “firebird” and no other normal user can read it. A typical installation will show the user and group as “firebird” (both read/write) with no “world access”. The directory path should also not be writeable by any user other than root or firebird, otherwise a malicious user could replace the security database with a compromised version.

- On Windows, similar remarks apply. isql should be run by a System Administrator and the Firebird security database only readable/writeable by the Firebird Server or a System Administrator.

2.4.1.4 Firebird 3 Alternative Security Databases

Firebird 3 can also assign an alternative security database on a per database basis. There can be as many alternative security databases as needed and they may be shared between databases and/or included within a user database. Each security database lists a set of users and their passwords and each may have a different SYSDBA password.

Alternative security databases are defined in the server's *database.conf* configuration file[12]. This replaced the *aliases.conf* file used prior to Firebird 3 and shares a similar syntax. In order to use an alternative security database, you must set up an alias for the database in the configuration file and then associate the alternative security database with the database alias. For example:

```
private.security.db = $(dir_secDb)/private.security.fdb
{
      RemoteAccess = false
}
employee = $(dir_sampleDb)/employee.fdb
{
      SecurityDatabase = private.security.db
}
```

assigns an alternative security database to the example employee database. The example is for a Debian derived system and locates the private security database in the same directory as the default security database[13]. The alternative security database is defined by an alias which also permits the prohibition of remote access to this database. Using an alias also simplifies administration when several databases share the same alternative security database. If the alias name is the same as the database's then the security database is embedded in the user database.

Note: there is a “chicken and egg” problem with the creation of an alternative security database. In the above example, if the employee database did not exist when the entry was created then it can only be created if the alternative security database already exists and contains a SYSDBA user. However, if this is the only database served by the security database then it is not possible to create the SYSDBA user (if one does not exist) unless the database exists (although it may be empty) and can be connected to by isql in embedded mode.

An alternative security database is created using isql in embedded mode e.g. on a Debian derived system:

```
isql-fb -user SYSDBA
SQL>create database 'private.security.db';
```

The above simply creates a regular Firebird Database. It is empty with no tables defined as yet. There is nothing to indicate its purpose;

Note: the location in which the database is created should be protected and not writeable by another other than a System Administrator or the Firebird Server. isql will need to be run by a System Administrator (e.g. root on a Posix system) and it may be necessary, after it has been created, to change the security database's ownership and file system access permissions so that it is exclusively available to the Firebird Server.

However, while the above creates the database, there is no connection to it. In order to create any user, including the SYSDBA user, you have to connect to a database which uses the newly created security database. This means that the using database must exist before its security database can be initialised. For example, following on from the above isql session:

```
SQL>connect employee;
SQL>create user SYSDBA password 'some-obscure-password';
SQL>commit;
SQL>quit;
```

Note: in order to create the SYSDBA, you have to connect to the database in embedded mode. Once the SYSDBA user has been created (it is automatically the superuser), you can now connect remotely as SYSDBA and create the other users. Creating the SYSDBA user also creates the tables needed to support the chosen user authentication plugin.

The CREATE USER syntax also allows the user authentication plugin to be specified and this may determine which authentication tables are created and how the user credentials are saved. For example:

```
SQL>create user SYSDBA password 'some-obscure-password' USING PLUGIN SRP;
```

If no plugin is specified then the default User Manager plugin is used (see 2.4.3).

2.4.1.5 Alternative Security Databases and the Services API

In Firebird 3 only the Services API can be used to manage users in the default security database. However, it is still necessary for there to be access from the Services API to databases that use an alternative security database. For example, when using the Services API for backup/restore.

In order to direct the Services API login to the appropriate security database, the Services Parameter Block (SPB) has a new item code: isc_spb_expected_db[14]. The value of this item is the name of a database which uses the alternative security database. The login will then be authenticated using the security database associated with the “expected db”.

Note: From IBX 2.2 onwards, the Services API components support a new login parameter (expected_db) which corresponds to the SPB isc_spb_expected_db item code.

User Mapping can also be used to provide access using the Services API to users authenticated using the default security database to databases that use an alternative security database (See 2.4.4).

2.4.1.6 Encryption of the Security Database

Firebird 3 does not provide an explicit mechanism to encrypt the security database. However, if the security database is included within a user database which is itself encrypted then the security database is also encrypted. This may well be desirable in environments where server security is under the control of the data owner (e.g. on cloud servers).

However, this means that the database encryption key has to be provided to the server prior to user authentication, given that the user authentication tables are in the encrypted database. On the other hand, a client should only pass the database encryption key to the server after it has authenticated the server. Otherwise, an attacker could set up a “man in the middle” attack to solicit the database encryption key.

This implies that there has to be a clear separation between client and server authentication, with server authentication taking place first, the encryption key passed to the server, and thus permitting client identification and authentication.

Note: the use of the term “identification”. By providing an encryption key, the client may have already be assumed to be an authorised user. However, which one? And which access rights should they be granted. That is why identification and authentication of that identity still need to take place.

2.4.2 Authentication

Prior to Firebird 3, only a single user authentication mechanism was supported by the Firebird Server. Firebird 3 supports multiple user authentication mechanisms through its plugin architecture, including:

- Legacy_Auth: user authentication compatible with older versions of Firebird and provided for backwards compatibility with older clients. This uses a simple password hash, is very insecure when compared with later versions, and its use should be discontinued as soon as possible.

- Srp: an implementation of the Secure Remote Password (SRP) protocol **[10]**.

- Win_Sspi: an interface to the Windows Security Support Provider Interface **[12]**, and is backwards compatible with Windows trusted authentication supported by Firebird 2.1 and 2.5.

Three parameters in the “firebird.conf” file configure the use of user authentication plugins:

- AuthServer: determines the plugins used by a Firebird Server for user authentication and the order in which they are tried.

- AuthClient: similarly determines the plugins used by a Firebird client to authenticate themselves to a server and the order in which they are tried.

- UserManager: is the plugin used for managing the security database. When more than one plugin is listed then the first is the default (used when no plugin is specified in the CREATE/ALTER/DROP USER DDL statements described below).

2.4.2.1 Legacy Authentication

Legacy Authentication is Firebird 2.x compatible and uses a SHA-1 based hash of the password, limited to first eight characters of the password given by the user. The authentication scheme is believed to be vulnerable to brute force attacks.

2.4.2.2 Secure Remote Password (SRP)

From Firebird 3 onwards, an implementation of the SRP-6 variant **[11]** of the Secure Remote Password (SRP) protocol **[10]** is provided as the default user authentication mechanism. As described in the RFC:

> (SRP) is suitable for negotiating secure connections using a user-supplied password, while eliminating the security problems traditionally associated with reusable passwords. This system also performs a secure key exchange in the process of authentication, allowing security layers (privacy and/or integrity protection) to be enabled during the session. Trusted key servers and certificate infrastructures are not required, and clients are not required to store or manage any long-term keys. SRP offers both security and deployment advantages over existing challenge-response techniques, making it an ideal drop-in replacement where secure password authentication is needed.

SRP-6/6a is described below.

SRP uses a derivative of the Diffie-Hellman key agreement scheme to generate a shared session key and its security depends upon the intractability of the Discrete Logarithm Problem. For each user recorded in the security database, SRP stores the triplet:

```
{ <username>, <password verifier>, <salt> }
```

where the “salt” is some random number and the password verifier is computed as:

```
x = H(<salt>, I, <raw password>))
<password verifier> = v = g^x % N
```

H is a Secure Hash Algorithm, and ',' represents concatenation. Firebird uses the 160 bit SHA-1 as its Secure Hash Algorithm. Note that the computation of the password verifier takes place in modulo number space (mod N).

Authentication proceeds by the client and server respectively passing to each other the user name (I) and salt (s). Each also generates a random number, and computing from these Diffie-Hellman exponentials A and B, respectively, which are also exchanged:

```
a = random()
A = g^a % N
                                  I, A →
                                                    b = random()
                                                    lookup(s,v) using I
                                                    B = (kv + g^b) % N
                                        ← s, B
```

k is a Multiplier parameter (k = H(N, g) in SRP-6a, k = 3 for legacy SRP-6. Firebird is understood to use SRP-6a.

Whilst A and B are exchanged in clear, the intractability of the Discrete Logarithm Problem means that it is not computationally feasible for an attacker to derived either a or b from observing the exchange of either A or B.

Both sides then compute a shared session key by[15]:

```
p = <raw password>                                  S = (A * v^u) ^ b % N
```

```
x = H(s, I, p))                                          K = H(S)
S = (B - kg^x) ^ (a + u * x) % N
K = H(S)
```

The parameter u = H(A , B).

A final verification exchange is used to demonstrate that both client and server have computed the same K and, by doing so, assert that the client has authenticated itself to the server and the server has authenticated itself to the client. This has been achieved without exchanging the password or a simple hash based on the password.

This is because:

1. If an incorrect password was used by the client, it would generate a very different value of K compared with that generated with the correct password. A server can assert that in order to generate K correctly, the user must have provided the correct password/username combination and is hence the identified user.

2. If an incorrect password verifier was used by the server, it would generate a very different value of K compared with that generated with the correct password. The client can assert that this must be the server that holds the user's authentication credentials and is hence the intended server.

The security of the exchange ultimately depends upon the user keeping their password secure and the server keeping the password verifiers secure.

- An attacker that steals a user's password can clear impersonate that user.

- Although it is computationally infeasible to determine the password from the password verifier, an attacker who was able to obtain a copy of the salt, user name, password verifier triplets could implement a limited server masquerade. For example, to solicit a database encryption key from a client.

SRP provides protection against a “man-in-the-middle” attack even when the password verifier is known to the attacker, given the difficulty of deriving either a or b from A or B.

The shared session key generated by SRP is used by Firebird when it needs to generate an ephemeral encryption key used for “over the wire encryption”.[16]

Although the RFC describes SRP as a “cryptographically strong network authentication mechanism”, the form of authentication offered is often known as “weak authentication”[17]. This is because authentication is based on a single proof – knowledge of a plain text password of up to 20 bytes in length[18]. The “cryptographically strong” tag applies to the difficulty of an attacker determining a password from observing the communication, brute force, or even by obtaining a copy of the authentication data held in the security database.

To qualify as “strong authentication”, there needs to be a third party involved in the proof and/or some form of two factor authentication.

2.4.2.3 Win_Sspi

Win_Sspi is not a user authentication scheme in itself, rather an interface between Firebird and the Windows Security Support Provider Interface (SSPI) API **[2]**. As documented by Microsoft:

> (The) Security Support Provider Interface (SSPI) allows an application to use various security models available on a computer or network without changing the interface to the security system. SSPI does not establish logon credentials because that is generally a privileged operation handled by the operating system.
>
> A security support provider (SSP) is contained in a dynamic-link library (DLL) that implements SSPI by making one or more security packages available to applications. Each security package provides mappings between the SSPI function calls of an application and the functions of an actual security model. Security packages support security protocols such as Kerberos authentication and LAN Manager.

Win_Sspi is only available on Windows Clients and Servers, and requires a common domain of trust. That is both client and server must share the same security support provider. Win_Sspi does not provide a means to generate a shared session key and hence may not be used with the current Firebird “over the wire” encryption mechanism.

When used, users are logged in to the Firebird database using their Windows logon user name. Win_Sspi implements windows trusted authentication and is backward compatible with 2.1 and 2.5 clients and servers running on windows.

In Firebird 2.1[19] and 2.5, the “Authentication” parameter could be set to:

- Native: Firebird's own authentication method
- Trusted: Windows authentication is used (equivalent to WinSspi).

  Note: If a local Adminstrator or a member of the built-in Domain Admins group connects to Firebird using trusted authentication, he/she will be connected as SYSDBA.
- Mixed: either of the above may be used – if a user and password are specified then “native” is implied.

In Firebird 2.5, automatic SYSDBA mapping was given more control when operating under “Trusted Authentication” and with a new DDL instruction:

```
ALTER ROLE RDB$ADMIN SET/DROP AUTO ADMIN MAPPING
```

That could configure this function on and off on a per server basis[20]. The RDB$ADMIN role is discussed later in 2.5.3.

2.4.2.4 SSL/TLS

Firebird does not currently implement SSL/TLS. However, the plugin architecture would allow third party development and deployment of an SSL/TLS provider to replace the existing “Remote” provider.

The benefits of using SSL/TLS would be:

- Server authentication using X.509 certificates
- Client authentication and identification using X.509 certificates.
- Built-in “over the wire encryption” using many different encryption protocols.
- May be integrated with SRP (see **[13]** and **[15]**).

X.509 certificates are signed with a trusted third party – the Certification Authority (CA). Authentication that involves a trusted third party is strong authentication, especially if an additional proof is provided, such as a password, when it may be considered a form of “two-factor authentication”. In the case of the client, the X.509 certificate may both identify and be limited to a single named user.

An SSL/TLS provider would always provide a server certificate to enable the client to authenticate the server. It could be configured to identify and authenticate clients either:

- By verifying an X.509 client certificate that include the client's user name, or
- By protecting a more traditional username/password exchange (with no client certificate required), or
- By demanding both of the above in order to gain a form of “two factor authentication”.

A benefit of SSL/TLS is that authentication using X.509 certificates could take place before any need to access the security database. It may thus permit a client to authenticate the server and to hence gain enough confidence to pass the database encryption key to the server prior to a need to access the security database for any user password verification. It is thus possible to place the user authentication tables (security database) in the encrypted database.

Unfortunately, while SRP has been integrated with SSL/TLS **[13]**, the implementation has been designed to optimise the protocol exchange and hence demands access to the password verifiers before the client has received the server certificate. Re-design would thus be necessary before it would be useful for protecting the exchange of a database encryption key.

It should also be noted that such a mode of operation would only be more secure than having a separate (in clear) security database if the server's private encryption key (which the certificate is used to verify) is itself passphrase encrypted when held on the server. This is because, an attacker who had access to an unencrypted security database is just as likely to have access to an unencrypted server key.

Passphrase protection of a server key is an approach often taken when X.509 certificates are installed on webservers in support of “https”. However, it does require that a means is provided for a System Administrator to access the server via a secure exchange in order to enter the passphrase and hence allow the encrypted key to be loaded into memory prior to use[21].

2.4.2.5 Summary

User Authentication in Firebird 3 is clearly a major improvement on earlier versions. SRP offers both user and server authentication is resistant to brute force attacks and man-in-the-middle attacks. As long as both a user's password and the security database are secured and are not available to an attacker, it is arguably as good as needed for most Firebird deployments.

The Win_Sspi approach used to be considered more secure than Firebird native authentication which is true when only legacy authentication is available. It is still likely to be more convenient for Windows users. However, as it does not appear to support “over the wire encryption”, database users that need to communicate with a server over unsecured data communications paths are likely to prefer SRP with over the wire encryption even when both ends are Windows systems.

However, there is a limitation with SRP when it comes to encrypted databases. If the reason for encrypting the database is that the server cannot be fully protected from an attacker then the SRP authentication data (password verifiers) in the security database are also vulnerable which, in turn,

risks that by taking a copy of the security database, an attacker could masquerade as the server and thereby solicit the database encryption key from a client authorised to access the database.

If the security database is also to be encrypted in order to mitigate such a risk, then it is necessary for the server to authenticate itself to the client prior to exchange of the database encryption key for the security database (probably embedded in the user's database). SSL/TLS could support such an approach, but not using the standard TLS-SRB approach. A Firebird specific design would be needed.

Alternatively, the security database might be separately encrypted using a passphrase provided in a similar manner to the passphrase for the server key.

2.4.3 User Management

Win_Sspi does not require the Firebird Server to keep track of users, as this is done externally by Windows security support providers. However, both Legacy_Auth and Srp maintain user credentials in the Firebird security database. In Firebird 3, each provides a User Management "plugin" for this purposes. In early versions, the Legacy_Auth User Manager was simply a built-in part of Firebird.

2.4.3.1 Firebird 2.1 and Earlier

Prior to Firebird 2.5, only the *gsec* utility and the Services API could be used to manage users. Either could be used to:

- List all users and their attributes
- Create a new user
- Modify an existing user's password or attributes
- Delete an existing user.

In this case, user attributes are limited: UID, GID, First Name, Middle Name and Last Name and are for information only. Each user is identified by a unique "User Name" and which is the key to the user's entry in the security database. A User Name once created cannot be modified.

Only the SYSDBA user could perform User Management activities.

2.4.3.2 Firebird 2.5

In addition to use of *gsec* and the Services API for User Management, Firebird 2.5 also introduced the CREATE/ALTER/DROP USER DDL statements **[6]**. These enable User Management from a Database Connection. The user must be logged in as SYSDBA or with the role RDB$ADMIN[22] (see 2.5.3). However, even a normal user can use ALTER USER to change their own password.

Firebird 2.5 also allows the "RDB$ADMIN" role to be granted to a normal user in respect of the security database itself. The user may then use CREATE/ALTER/DROP USER DDL statements or the Services API to perform User Management in addition to the SYSDBA.

The new DDL statements are defined as:

2.4.3.2.1 CREATE USER

Description: Creates a Firebird user account.

Syntax:

```
CREATE USER username PASSWORD 'password'
   [FIRSTNAME 'firstname']
   [MIDDLENAME 'middlename']
   [LASTNAME 'lastname']
   [GRANT ADMIN ROLE]
```

GRANT ADMIN ROLE gives the new user the RDB$ADMIN role in the security database. This allows them to manage user accounts, but doesn't give them any special privileges in regular databases.

2.4.3.2.2 ALTER USER

Description: Alters details of a Firebird user account. This is the only account management statement that can also be used by non-privileged users, in order to change their own account details.

Syntax:

```
ALTER USER username
   [PASSWORD 'password']
   [FIRSTNAME 'firstname']
   [MIDDLENAME 'middlename']
   [LASTNAME 'lastname']
   [{GRANT|REVOKE} ADMIN ROLE]

-- At least one of the optional parameters must be present.
-- GRANT/REVOKE ADMIN ROLE is reserved to privileged users.
```

2.4.3.2.3 DROP USER

Description: Removes a Firebird user account.

Syntax:

```
DROP USER username
```

2.4.3.3 Firebird 3

The weakness in Firebird 2.5, from the point of view of User Management from a database connection is that there was no way to list the existing users. The Services API has to be used for this purpose. With the introduction of alternative security databases and the restriction of the Services API to the default security database, a new mechanism was needed to list the users.

The SEC$USERS virtual table was introduced in Firebird 3[23] and lists the users, and their attributes, in the security database associated with the database. If the user is logged in as SYSDBA or with the RDB$ADMIN role then all users are listed. Otherwise, the list is restricted to the current user. The list is also restricted to the users managed by each of the User Managers listed in the “firebird.conf” configuration file.

Note: the user list can still be displayed using the services API. However, the list is limited to users managed using the default user manager.

2.4.3.3.1 DDL Extensions

Firebird 3 has also extended[24] the CREATE/ALTER/DROP USER syntax introduced in Firebird 2.5:

```
CREATE USER username [ options_list ] [ TAGS ( tag [, tag [, tag ...]] ) ]
ALTER USER username [ SET ] [ options_list ] [ TAGS ( tag [, tag [, tag ...]]
) ]
ALTER CURRENT USER [ SET ] [ options_list ] [ TAGS ( tag [, tag [,
tag ...]] ) ]
CREATE OR ALTER USER username [ SET ] [ options ] [ TAGS ( tag [, tag [,
tag ...]] ) ]
DROP USER username [ USING PLUGIN plugin_name ]
```

OPTIONS is a (possibly empty) list with the following options:

```
PASSWORD 'password'
FIRSTNAME 'string value'
MIDDLENAME 'string value'
LASTNAME 'string value'
ACTIVE
INACTIVE
USING PLUGIN plugin_name
```

Each TAG may have one of two forms:

```
TAGNAME = 'string value'
```

or the DROP *TAGNAME* tag form to remove a user-defined attribute entirely:

```
DROP TAGNAME
```

The Active/Inactive option allows a user to be disabled rather than deleted. The “USING PLUGIN” clause allows multiple authentication plugins to be used for the same security database. The name referred to is the User Manager plugin name and not the authentication plugin name.

Note: For Srp, both the User Manager and the authentication plugin have the same name “Srp”. For Legacy Authentication the names are respectively: Legacy_UserManager and Legacy_Auth.

2.4.3.3.2 User Tags

User tags are additional attributes in “key=value” format that are available for use for unspecified purposes. They are managed using the CREATE/ALTER USER DDL statements and those specified for a given user can be read using the SEC$USER_ATTRIBUTES virtual table.

Note: The legacy user attributes UID and GID and copied to a Firebird 3 security database as tags when a Firebird 2 security database is upgraded[25].

2.4.4 Mapping Users between Security Contexts

Firebird 3 introduces the concept of “mapping rules”. These have two purposes:

a) To manage the mapping between users defined by an external user authentication mechanism (e.g. a Windows Security Support Provider) into Firebird users and roles, and

b) To allow users authenticated using one user authentication plugin/security database to be mapped to users and roles in a database configured to use a different user authentication plugin/security database.

In the second case, such mappings are limited to using the DML “EXECUTE STATEMENT ON EXTERNAL DATA SOURCE[26]” and the Services API and do not work with connections to databases.[27] This applies (e.g.) to mapping rules that allow a user (e.g. SYSDBA) authenticated in (e.g. the default security database) to be mapped to the SYSDBA user in a database using an alternative security database. Such a capability map be used to simplify overall database administration with a unified SYSDBA login.

Mapping rules are created by DDL statements executed in the database to which they apply. They are defined in **[2]** as:

```
{CREATE | ALTER | CREATE OR ALTER} [GLOBAL] MAPPING name
  USING {
    PLUGIN name [IN database] | ANY PLUGIN [IN database | SERVERWIDE] |
      MAPPING [IN database] | '*' [IN database]}
  FROM {ANY type | type name}
  TO {USER | ROLE} [name]
  --
DROP [GLOBAL] MAPPING name
```

- A Global Mapping applies to all databases that use the current security database. A local mapping applies only to the current database.

- The Using clause is used to identify the source of the mapping defined as an authentication method (plugin) and the security database in which the source of the mapping is defined. The “DATABASE” must conform with syntax rules for an SQL identifier.

  Note: 'security.db' is conventionally defined in the Firebird databases.conf file to identify the default security database. It must be enclosed in double quotes, in the above example, as it includes the '.' character.

- The From clause identifies the type and name of the subject that is being mapped into the local database. The “type” is a subject defined by the plugin (with SRP “user” is a type name)

- The To clause identifies what is being mapped to. Either a user or a role.

An example of a mapping rule is:

```
CREATE MAPPING DEF_SYSDBA
  USING PLUGIN SRP IN "security.db"
  FROM USER SYSDBA
  TO USER;
```

which should allow the server's SYSDBA (from the main security database) to access the current database using the Services API. (Assume that the database is using a non-default security database). Alternatively,

```
CREATE GLOBAL MAPPING TRUSTED_AUTH
  USING PLUGIN WIN_SSPI
  FROM ANY USER
  TO USER;
```

enables the use of Windows trusted authentication in all databases that use the current security database, while

```
CREATE MAPPING WIN_ADMINS
  USING PLUGIN WIN_SSPI
  FROM Predefined_Group
  DOMAIN_ANY_RID_ADMINS
  TO ROLE RDB$ADMIN;
```

enables SYSDBA-like access for windows administrators in current database.

Note: The group DOMAIN_ANY_RID_ADMINS does not exist in Windows, but is instead added by the `win_sspi` plug-in to provide exact backwards compatibility.

2.5 Roles

Firebird implements “Roles” as specified in the SQL standard, where a role may be considered to be a named user group. Access rights can be assigned (granted) to roles in the same way as they are assigned to users (see 2.6). A user can be allowed to use (granted) many different roles and may assume a role they may use either when they login or, from Firebird 3 onwards, at any time during a database connection.

Once a user assumes a role they are implicitly granted all access rights assigned to the role in addition to any access rights they are assigned individually.

Roles can be viewed as a means of greatly simplifying user administration. A Database Architect should determine the different types of user that may access the database and the needs of each such user group. A role should be created for each such user group and appropriate access rights assigned to the role.

When a user is given access to a database, rather than having to work out which tables, views and stored procedures they need to access, all the database administrator need to do is to identify the most appropriate role(s) for them and grant them use of those roles.

2.5.1 Creating a Role

The creation of role objects should properly be considered as part of the database schema defintion rather than a day to day Database Administration task. A role[28] is created using the DDL Statement:

```
Create Role <rolename>;
```

and deleted using:

```
Drop Role <roleName>;
```

2.5.2 Users and Roles

A user is granted use of a role using the statement:

```
Grant <rolename> to <username>;
```

and the use of a role revoked using the statement:

```
Revoke <rolename> from <username>;
```

Note: revoking the granted roles from a given user tidily removes their access rights to the objects in the database. However, this only affects the current database and not any backups. This is because the privilege to use a given role is held in the database itself. One of the tasks of Database Administrator is to keep track of when privileges are revoked so that should it be necessary to restore a database backup, all rights revoked since that backup are, once again, revoked.

2.5.3 The RDB$ADMIN role

The RDB$ADMIN role was introduced in Firebird 2.5 **[5]** in order to enable the transfer of "SYSDBA privileges to another user. Any user, when granted the role in a particular database, acquires SYSDBA-like rights when attaching to that database with the RDB$ADMIN role specified."

The RDB$ADMIN role does not have to be created and may be considered as being "built-in". It is granted and revoked in the same way as any other role. Only the SYSDBA or a user assuming the RDB$ADMIN role can grant or revoke this role.

2.5.3.1 With Trusted Authentication

Firebird 2.5 also allowed Windows Administrators logged in using trusted authentication (see 2.4.2.3) to automatically be granted the RDB$ADMIN role i.e.

```
ALTER ROLE RDB$ADMIN SET AUTO ADMIN MAPPING;
ALTER ROLE RDB$ADMIN DROP AUTO ADMIN MAPPING;
```

are used to respectively to enable the capability and to disable it at the server level.

Note: in Firebird 2.1, the RDB$ADMIN role did not exist and Windows Administrators logged in using trusted authentication automatically become the SYSDBA.

Firebird 3 has superseded auto admin mapping for Windows Administrators with the more general concept of user mappings (see 2.4.4).

2.5.3.2 Use with the Security Database

Firebird 2.5 also allowed the RDB$ADMIN role to extend to the security database – although this is more like an administrator privilege rather than a role and is granted/revoked using an entirely different syntax. That is using the CREATE/ALTER/USER statement (see 2.4.3.2).

When a user has been granted the "ADMIN ROLE" using (e.g.) and ALTER USER statement, they may use either the Services API or the CREATE/ALTER/USER statements to manage users.

The capability remains unchanged in Firebird 3.

2.5.3.3 RDB$ADMIN and the Services API

As described in the "Other Services API Additions" section of **[8]**, the Services API was also extended by Firebird 2.5 to:

a) Allow the auto mapping of the RDB$ADMIN role to be enabled/disabled through the Services API, and

b) Allow the grant/revoke of the security database admin privilege to a user to be performed.

2.5.4 Assuming a Role

Prior to Firebird 3, it was only possible to assume a role when a user logged into a database. Firebird 3 iintroduces the SET ROLE statement which allows a user to assume, post login, a role that has been previously granted to them. The statement syntax is:

```
SET ROLE <rolename>
```

2.5.5 The Trusted Role

2.6 Access Rights

2.7 Summary

3

Legacy User Management and IBX

4

Firebird 3 User Management and IBX

1 This is described in the section “Security->Over the Wire Connection Encryption” in the Firebird 3.0.2 Release Notes https://www.firebirdsql.org/file/documentation/release_notes/html/en/3_0/rlsnotes30.html. Note that an external “plugin” is used which permits extension to encryption algorithms other than the default ARC4. An emphemeral session key needs also to be agreed and this is usually a function of the user authentication mechanism.

2 Metadata access controls where introduced in Firebird 3. Prior releases allowed any user to create an object, whilst limiting changes to an object's owner or a “superuser”.

3 The Firebird 2.5 release notes describe the RDB$ADMIN role as something that can be granted to a user on a per database basis and/or to the security database. At the database level, it is granted the same as any other role using a GRANT DDL statement (e.g. GRANT RDB$ADMIN TO ALICE). However, an entirely different syntax is used for the security database (e.g. ALTER USER ALICE GRANT ADMIN ROLE”). See RDB$ADMIN Role in Firebird 2.5.8 Release Notes https://www.firebirdsql.org/file/documentation/release_notes/html/en/2_5/rlsnotes25.html

4 Prior to Firebird 3, the System Tables used to hold a database's metadata were both publicly readable and writeable by a System Administrator. In Firebird 3, they become read only, with explicit permissions controlling which users can update the metadata and which metadata objects they can update (through DDL statements). (e.g. making “Create Table” and “Create Procedure” separate assignable privileges”. See “User Privileges for Metadata Changes” Firebird 3.0.2 Release Notes https://www.firebirdsql.org/file/documentation/release_notes/html/en/3_0/rlsnotes30.html.

5 This is an often overlooked aspect of file system security. If the file system path can be modified then this introduces all sorts of ways in which an attacker can make mischief. For example, by diverting the path to the security database to point to a compromised version.

6 Prior to Firebird 3, this file was called “aliases.conf”.

7 It should be recalled that a major DAC requirement is that “The TCB shall protect authentication data so that it cannot be accessed by any unauthorized user”. It is very easy to overlook the importance of protecting the security database and, in the absence of database encryption, that file system protections have to be relied upon to do this. This includes both limiting read and write access to authorised users only and ensuring that the database's file system path is protected from unauthorised modification. Otherwise a malicious user could replace the security database with a compromised version. All directories in the security database's file system path must restrict write access to authorised users only.

8 Firebird's vulnerability to brute force attacks has been significantly reduced over successive releases. For a discussion of this subject see the “Details of the Security Changes in Firebird 2” in the Firebird 2.1.7 Release Notes https://www.firebirdsql.org/file/documentation/release_notes/html/rlsnotes217.html

9 This subject is discussed at length in the release notes. See section “Details of the Security Changes in Firebird 2.0” in Firebird 2.0.7 Release Notes https://www.firebirdsql.org/file/documentation/release_notes/html/rlsnotes207.html.

10 If this feature is used then the DBA needs to be confident that the user authentication tables are adequately protected from both read and write access by normal users. This includes preventing normal users from making a remote backup of the database (including authentication tables). See “Location of User Lists” in Firebird 3.0.2 Release Notes https://www.firebirdsql.org/file/documentation/release_notes/html/en/3_0/rlsnotes30.html

11 In this case, the client will need to provide the encryption key to the server before the client is authenticated. However, in order to avoid keys being inadvertently disclosed to an attacker, the user authentication protocol must permit the server to be authenticated to the client prior to communication of the encryption key.

12 See section on “Creating an Alternative Security Database” Firebird 3.0.2 Release Notes https://www.firebirdsql.org/file/documentation/release_notes/html/en/3_0/rlsnotes30.html.

13 For a list of all parameters available for use in *databases.conf* and the file's syntax see the section “Per-database Configuration” in Firebird 3.0.2 Release Notes https://www.firebirdsql.org/file/documentation/release_notes/html/en/3_0/rlsnotes30.html.

14 This is described in the file “doc/README.services_extension” provided with the Firebird 3 source code.

15 There is a different between the SRP-6 algorithm described in Wu's paper and that presented on the SRP website. Specifically, in the original RFC and Wu's SRP-6 paper, the parameter x = H(s,I,P) while on the SRP website it is given as x =H(s,p) i.e. omitting the user name. Wu's paper is used as the source of the description.

16 In the "Over the wire" Connection Encryption section of the Firebird 3 release notes, it is stated that “RC4 uses a symmetric key which can have any length, while the key produced by SRP has a length of 20 bytes. That key is a SHA-1 hash on SRP's session key and some other SRP-related things, such as user name”. See Firebird 3.0.2 Release Notes https://www.firebirdsql.org/file/documentation/release_notes/html/en/3_0/rlsnotes30.html

17 A useful definition of Weak Authentication is provided by Jari Arkko, Pekka Nikander as “cryptographic authentication between previously unknown parties without relying on trusted third parties.” (https://www.ietf.org/proceedings/57/slides/enroll-4/enroll-4.ppt)

18 The SRP authentication plugin is stated as having an effective password length of 20 bytes – although as International Character Sets are also supported by Firebird 3 (e.g. UTF8), the actual effective password size in characters is not easy to quantify. It is also not clear as to where this restriction comes from. See “Chapter 3 → Plugins Q&A” Firebird 3.0.2 Release Notes https://www.firebirdsql.org/file/documentation/release_notes/html/en/3_0/rlsnotes30.html

19 See Configuration Parameter “Authentication” in the Firebird 2.1.7 Release Notes https://www.firebirdsql.org/file/documentation/release_notes/html/rlsnotes217.html.

20 See “Global Admin Privileges for Windows Administrators” in the Firebird 2.5.8 Release Notes https://www.firebirdsql.org/file/documentation/release_notes/html/en/2_5/rlsnotes25.html.

21 This is a common feature with the Apache webserver. When it uses an encrypted server key, the server must be manually started by a System Administrator who is then prompted for the passphrase. Once the key is loaded into memory, it is available for use until the server is stopped.

22 This is also described in the file “doc/README.services_extension” provided with the Firebird 3 source code.

23 See section “New System Tables” in Firebird 3.0.2 Release Notes https://www.firebirdsql.org/file/documentation/release_notes/html/en/3_0/rlsnotes30.html.

24 See section “Security->SQL Features for Managing Access” in Firebird 3.0.2 Release Notes https://www.firebirdsql.org/file/documentation/release_notes/html/en/3_0/rlsnotes30.html

25 The upgrade procedure is described in the section “Upgrading a v.2.x Security Database” in the Firebird 3.0.2 Release Notes https://www.firebirdsql.org/file/documentation/release_notes/html/en/3_0/rlsnotes30.html.

26 This was introduced in Firebird 2.5. See section “New Extensions to EXECUTE STATEMENT” in the Firebird 2.5.8 Release Notes https://www.firebirdsql.org/file/documentation/release_notes/html/en/2_5/rlsnotes25.html.

27 The Firebird 3 Release Notes make clear in that mappings are there to support (a) when EXECUTE STATEMENT ON EXTERNAL DATA SOURCE requires some data exchange between database clusters using different security databases, (b) when server-wide SYSDBA access to databases is needed from other clusters, using services, and (c) with Windows trusted authentication, to relate the Windows security database with Firebird's. However, while there is no explicit statement that database connections are not supported, testing shows that while mapping of Firebird native users works with the Services API, it does not work with database connections.

28 Rolenames follow the same naming rules as user names. However, “It is advisable to make the name of a role unique among user names as well. The system will not prevent the creation of a role whose name clashes with an existing user name but, if it happens, the user will be unable to connect to the database.” This is a quote from the “DDL Statements” section in the Firebird 2.5 Language Reference