SYNOPSIS

We study the problem of key establishment for secure many-to-many communications. The problem is inspired by the proliferation of large-scale distributed file systems supporting *parallel access* to multiple storage devices. Our work focuses on the current Internet standard for such file systems, *i.e.*, parallel Network File System (pNFS), which makes use of Kerberos to establish parallel session keys between clients and storage devices. Our review of the existing Kerberos-based protocol shows that it has a number of limitations: (i) a metadata server facilitating key exchange between the clients and the storage devices has heavy workload that restricts the scalability of the protocol; (ii) the protocol does not provide forward secrecy; (iii) the metadata server generates itself all the session keys that are used between the clients and storage devices, and this inherently leads to key escrow. In this paper, we propose a variety of authenticated

key exchange protocols that are designed to address the above issues. We show that our protocols are capable of reducing up to approximately 54% of the workload of the metadata server and concurrently supporting forward secrecy and escrow-freeness. All this requires only a small fraction of increased computation overhead at the client.

1. INTRODUCTION

In a parallel file system, file data is distributed across multiple storage devices or nodes to allow concurrent access by multiple tasks of a parallel application. This is typically used in large-scale cluster computing that focuses on high performance and reliable access to large datasets. That is, higher I/O bandwidth is achieved through concurrent access to multiple storage devices within large compute clusters; while data loss is protected through data mirroring using fault-tolerant striping algorithms. Some examples of high performance parallel file systems that are in production use are the IBM General Parallel File System (GPFS), Google File System (GoogleFS), Lustre, Parallel Virtual File System (PVFS), and Panasas File System; while there also exist research projects on distributed object storage systems such as Usra Minor, Ceph, XtreemFS, and Gfarm. These are usually required for advanced scientific or data-intensive applications such as, seismic data processing, digital animation studios, computational fluid dynamics, and semiconductor manufacturing. In these environments, hundreds or thousands of file system clients share data and generate very high aggregate I/O load on the file system supporting petabyte- or terabyte-scale storage capacities.

Independent of the development of cluster and high performance computing, the emergence of clouds and the MapReduce programming model has resulted in file systems such as the Hadoop Distributed File System (HDFS), Amazon S3 File System, and Cloud- Store. This, in turn, has accelerated the wide-spread use of distributed and parallel computation on large datasets in many organizations. Some notable users of the HDFS include AOL, Apple, eBay, Facebook, Hewlett-Packard, IBM, LinkedIn, Twitter, and Yahoo!

ABOUT THE SOFTWARE

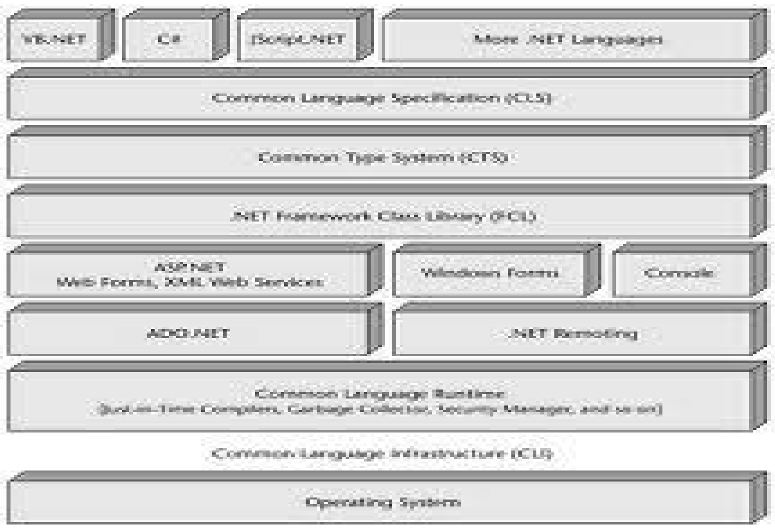
WHAT IS “.NET”?

Microsoft .net is a set of micro soft software technologies for rapidly building and integrating xml web services, micro soft windows-based applications, and web solutions. The .net framework is a language-neutral platform for writing programs that can easily and securely interoperate. There’s no language barrier with .net: there are numerous languages available to the developer including managed c++, c#, visual basic and java script.

The .net framework provides the foundation for components to interact seamlessly, whether locally or remotely on different platforms. It standardizes common data types and communications protocols so that components created in different languages can easily interoperate. “.net” is also the collective name given to various software components built upon the .net platform. These will be both products (visual studio.net and windows.net server, for instance) and services (like passport, .net my services, and so on).

The .NET Framework

Microsoft designed C# from the ground up to take advantage of its new .NET Framework. Because C# is a player in this new .NET world, you should have a good understanding of what the .NET Framework provides and how it increases your productivity.


VB.NET
C#
JScript.NET
More .NET Languages
Common Language Specification (CLS)
Common Type System (CTS)
.NET Framework Class Library (FCL)
ASP.NET Web Forms, XML Web Services
Windows Forms
Console
ADO.NET
.NET Remoting
Common Language Runtime (Just-in-Time Compilers, Garbage Collector, Security Manager, and so on)
Common Language Infrastructure (CLI)
Operating System

The .NET Framework is made up of four parts, as shown in the Common Language Runtime, a set of class libraries, a set of programming languages, and the ASP.NET environment. The .NET Framework was designed with three goals in mind. First, it was intended to make Windows applications much more reliable, while also providing an application with a greater degree of security. Second, it was intended to simplify the development of Web applications and services that not only work in the traditional sense, but on mobile devices as well. Lastly, the framework was designed to provide a single set of libraries that would work with multiple languages. The following sections examine each of the .NET Framework components.

Web Development:

The .NET Framework was designed with one thing in mind: to fuel Internet development. This new fuel to add to Internet development is called *Web Services*. You can think of Web Services as a Web site that interacts with programs, rather than people. Instead of delivering Web pages, a Web Service takes a request formatted as XML, performs a particular function, and then returns a response to the requester as an XML message.

Note XML or eXtensible Markup Language is a self describing language much like that of HTML. XML on the other hand has no predefined tags thus allowing it great flexibility in representing a wide variety of objects.

A typical application for a Web Service would be to sit as a layer on top of a corporate billing system. When a user surfing the Web purchases products from your Internet site, the purchase information is then sent to the Web Services, which totals all the products, adds a record to the accounts receivable database, and then returns a response with an order confirmation number. Not only can this Web Service interact with Web pages, it can interact with other Web Services, such as a corporate accounts payable system.

In order for the Web Service model to survive the natural evolution of programming languages, it must include much more than a simple interface to the Web. The Web service model also includes protocols that enable applications to find Web Services available across a LAN or the Internet. This protocol also enables the application to explore the Web Service and determine how to communicate with it, as well as how to exchange information. To enable Web Service discovery, the Universal Discovery, Description and Integration (UDDI) was established. This allows Web Services to be registered and searched, based on key information such as company name, type of service, and geographic location.

Application Development

Aside from Web development, you can still build traditional Windows applications with the .NET Framework. Windows applications created with the .NET Framework are based upon *Windows Forms*. These Windows Forms are somewhat of a crossbreed between Visual Basic 6 forms and the forms of Visual C++. Though forms look the same as their predecessors, they are completely object-oriented and class-based, much like form objects in the Microsoft Foundation Class. These new Windows Forms now support many classic controls found in Visual Studio, such as the Button, TextBox, and Label, as well as ActiveX controls. Aside from the traditional controls, new components such as PrintPreview, LinkLabel, ColorDialog, and OpenFileDialog are also supported.

Building applications with .NET also provides you with many enhancements not found in other languages, such as security. These security measures can determine whether an application can write or read a disk file. They also enable you to embed digital signatures into
the application to ensure that the application was written by a trusted source. The .NET Framework also enables you to embed component information, and version information, within the actual code. This makes it possible for software to install on demand, automatically, or with no user intervention at all. Together, all of these features greatly reduce support costs within the enterprise.

Common Language Runtime

Programming languages usually consist of both a compiler and a runtime environment. The compiler turns the code that you write into executable code that can be run by users. The runtime environment provides a set of operating system services to your executable code. These services are built into a runtime layer so that your code does not need to worry about the low-level details of working with the operating system. Operations such as memory management and file I/O are good examples of services that might be provided by a runtime environment. Before .NET came along, each language shipped with its own runtime environment. Visual Basic shipped with a runtime called MSVBVM60.DLL. Visual C++ shipped with a DLL called MSVCRT.DLL. Each of these runtime modules provided a set of low-level services to code that developers wrote. Developers would write code and then build that code with the appropriate runtime in mind. The executable code would ship with the runtime, which would be installed on a user's machine if it weren't already present.

The main problem with these runtime environments is that they were designed for use with a single language. The Visual Basic runtime provided nice features for operations like working with memory and launching COM objects, but these features were only available to Visual Basic users. Developers using Visual C++ could not use the features of the Visual Basic runtime. Visual C++ users had their own runtime, with its own long list of features, but those features were unavailable to Visual Basic users. This "separate runtime" approach prevented languages from working together seamlessly.

It's not possible, for example, to grab some memory in a piece of Visual Basic code and then hand it off to a piece of Visual C++ code, which frees the memory. The different runtimes implement their own feature set in their own way. The feature sets of the various runtimes are inconsistent. Even features that are found in more than one runtime are implemented in different ways, making it impossible for two pieces of code written in different languages to work together.

One of the design goals of the .NET Framework was to unify the runtime engines so that all developers could work with a single set of runtime services. The .NET Framework's solution is called the *Common Language Runtime (CLR)*. The CLR provides capabilities such as memory management, security, and robust error-handling to any language that works with the .NET Framework. Thanks to the CLR, all .NET languages can use a variety of runtime services without developers worrying about whether their particular language supports a runtime feature.

The CLR also enables languages to interoperate with one another. Memory can be allocated by code written in one language — Visual Basic .NET, for instance — and can be freed by code written in another language, say, C#. Similarly, errors can be raised in one language and processed in another language.

.NET Class Libraries

Developers like to work with code that has already been tested and shown to work, such as the Win32 API and the MFC Class libraries. Code re-use has long been the goal of the software development community. However, the practicality of code re-use has not lived up to expectations. Many languages have had access to bodies of pre-tested, ready-to-run code. Visual C++ has benefited from class libraries such as the Microsoft Foundation Classes (MFC), which enabled C++ developers to build Windows applications quickly, and the Active Template Library (ATL), which provided support for building COM objects. However, the languagespecific nature of these libraries has made them unavailable for use in other languages.

Visual Basic developers are locked out of using ATL when building their COM objects. The .NET Framework provides many classes that help developers re-use code. The .NET class libraries contain code for programming topics such as threading, file I/O, database support, XML parsing, and data structures, such as stacks and queues. Best of all, this entire class library is available to any programming language that supports the .NET Framework. Thanks to the CLR, any .NET language can use any class in the .NET class library. Because all languages now support the same runtime, they can re-use any class that works with the .NET Framework. This means that any functionality available to one language will also be available to any other .NET language.

The class library re-use picture painted by the .NET Framework gets even better when you realize that re-use extends to your code, not just code that Microsoft ships with .NET. The code that Microsoft ships in the .NET class library code base is architecturally no different from the code you write. The Microsoft code is simply code that was written using a language supported by .NET and built using a .NET development tool. This means that Microsoft is using the same tools that you will use to write your code. You can write code that can be used in other

.NET languages, just as Microsoft has with its class library. The .NET Framework enables you to write code in C#, for example, and hand it off to Visual Basic .NET developers, who can use your compiled code in their applications.

.NET Programming Language

The .NET Framework provides a set of tools that help you build code that works with the .NET Framework. Microsoft provides a set of languages that are already ".NET-compatible". C# is one of those languages. New versions of Visual Basic and Visual C++ have also been created to take advantage of the .NET Framework, with a version of Jscript.NET on the way. The development of .NET-compatible languages is not restricted to Microsoft. The .NET group at Microsoft has published documentation showing how language vendors can make their languages work with .NET, and vendors are making languages such as COBOL and Perl compatible with the .NET Framework. There are currently 20 or more languages in the works from third party vendors and institutions that plug into the .NET Framework.

Introducing C#

C#, the new language introduced in the .NET Framework, is derived from C++. However, C# is a modern, objected-oriented (from the ground up) type-safe language.

Language features

The following sections take a quick look at some of the features of the C# language. If some of these concepts don't sound familiar to you, don't worry. All of them are covered in detail in later chapters.

Classes

All code and data in C# must be enclosed in a class. You can't define a variable outside of a class, and you can't write any code that's not in a class. Classes can have *constructors,* which execute when an object of the class is created, and a *destructor,* which executes when an object of the class is destroyed. Classes support single inheritance, and all classes ultimately derive from a base class called *object*. C# supports versioning techniques to help your classes evolve over time while maintaining compatibility with code that uses earlier versions of your classes.

Data types

C# lets you work with two types of data: value types and reference types. *Value types* hold actual values. *Reference types* hold references to values stored elsewhere in memory. Primitive types such as char, int and float, as well as enumerated values and structures, are value types. Reference types hold variables that deal with objects and arrays. C# comes with predefined reference types (object and string), as well as predefined value types (sbyte, short, int, long, byte, ushort, uint, ulong, float, double, bool, char, and decimal). You can also define your own value and reference types in your code. All value and reference types ultimately derive from a base type called object.

C# allows you to convert a value of one type into a value of another type. You can work with both *implicit* conversions and *explicit* conversions. Implicit conversions always succeed and don't lose any information (for example, you can convert an int to a long without losing any data because a long is larger than an int). Explicit conversions may cause you to lose data (for example, converting a long into an int may result in a loss of data because a long can hold larger values than an int). You must write a cast operator into your code to make an explicit conversion happen.

You can work with both one-dimensional and multidimensional arrays in C#. Multidimensional arrays can be rectangular, in which each of the arrays has the same dimensions, or jagged, in which each of the arrays has different dimensions. Classes and structures can have data members called *properties* and *fields*. *Fields* are variables that are associated with the enclosing class or structure. You may define a structure called Employee, for example, that has a field called Name. If you define a variable of type Employee called CurrentEmployee, you can retrieve the employee's name by writing CurrentEmployee.Name. *Properties* are like fields, but enable you to write code to specify what should happen when code accesses the value. If the employee's name must be read from a database, for example, you can write code that says, "when someone asks for the value of the Name property, read the name from the database and return the name as a string."

Functions

A function is a callable piece of code that may or may not return a value to the code that originally called it. An example of a function would be the FullName function shown earlier, in this chapter, in the Family class. A *function* is generally associated to pieces of code that return information whereas a *method* generally does not return information. For our purposes however, we generalize and refer to them both as functions. Functions can have four kinds of parameters:

• Input parameters have values that are sent into the function, but the function cannot change those values.
• Output parameters have no value when they are sent into the function, but the function can give them a value and send the value back to the caller.
• Reference parameters pass in a reference to another value. They have a value coming in to the function, and that value can be changed inside the function.
• Params parameters define a variable number of arguments in a list.

C# and the CLR work together to provide automatic memory management. You don't need to write code that says "allocate enough memory for an integer" or "free the memory that this object was using." The CLR monitors your memory usage and automatically retrieves more when you need it. It also frees memory automatically when it detects that it is no longer being used (this is also known as Garbage Collection). C# provides a variety of operators that enable you to write mathematical and bitwise expressions. Many (but not all) of these operators can be redefined, enabling you to change how the operators work.

C# supports a long list of statements that enable you to define various execution paths within your code. Flow control statements that use keywords such as if, switch, while, for, break and continue enable your code to branch off into different paths, depending on the values of your variables. Classes can contain code and data. Each class member has something called an *accessibility scope,* which defines the member's visibility to other objects. C# supports public, protected, internal, protected internal, and private accessibility scopes.

Variables

Variables can be defined as constants. *Constants* have values that cannot change during the execution of your code. The value of pi, for instance, is a good example of a constant, because its value won't be changing as your code runs. *Enum type declarations* specify a type name for a related group of constants. For example, you could define an enum of Planets with values of Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune and Pluto, and use those names in your code. Using the enum names in code makes code more readable than if you used a number to represent each planet.

C# provides a built-in mechanism for defining and handling events. If you write a class that performs a lengthy operation, you may want to invoke an event when the operation is completed. Clients can subscribe to that event and catch the event in their code, which enables them to be notified when you have completed your lengthy operation. The event handling mechanism in C# uses *delegates*, which are variables that reference a function. Note An event handler is a procedure in your code that determines the actions to be performed when an event occurs, such as the user clicking a button.

If your class holds a set of values, clients may want to access the values as if your class were an array. You can write a piece of code called an *indexer* to enable your class to be accessed as if it were an array. Suppose you write a class called Rainbow, for example, that contains a set of the colors in the rainbow. Callers may want to write MyRainbow[0] to retrieve the first color in the rainbow. You can write an indexer into your Rainbow class to define what should be returned when the caller accesses your class, as if it were an array of values.

Interfaces

C# supports *interfaces*, which are groups of properties, methods, and events that specify a set of functionality. C# classes can implement interfaces, which tells users that the class supports the set of functionality documented by the interface. You can develop implementations of interfaces without interfering with any existing code, which minimizes compatibility problems. nce an interface has been published, it cannot be changed, but it can evolve through inheritance.

C# classes can implement many interfaces, although the classes can only inherit from a single base class.Let's look at a real-world example that would benefit from interfaces to illustrate its extremely positive role in C#. Many applications available today support add-ins. Assume that you have created a code editor for writing applications. This code editor, when executed, has the capability to load add-ins. To do this, the add-in must follow a few rules. The DLL add-in must export a function called CEEntry, and the name of the DLL must begin with CEd. When we run our code editor, it scans its working directory for all DLLs that begin with CEd. When it finds one, it is loaded; and then it uses the GetProcAddress to locate the CEEntry functionwithin the DLL, thus verifying that you followed all the rules necessary to create an add-in.

This method of creating and loading add-ins is very burdensome because it burdens the code editor with more verification duties than necessary. If an interface were used in this instance, your add-in DLL could have implemented an interface, thus guaranteeing that all necessary methods, properties, and events were present with the DLL itself, and functioning as documentation specified.

Attributes

Attributes declare additional information about your class to the CLR. In the past, if you wanted to make your class self-describing, you had to take a disconnected approach in which the documentation was stored in external files such as IDL or even HTML files. Attributes solve this problem by enabling you, the developer, to bind information to classes — any kind of information. For example, you can use an attribute to embed documentation information into a class. Attributes can also be used to bind runtime information to a class, defining how it should act when used. The possibilities are endless, which is why Microsoft includes many predefined attributes within the .NET Framework.

Compiling C#

Running your C# code through the C# compiler produces two important pieces of information: code and metadata. The following sections describe these two items and then finish up by examining the binary building block of .NET code: the assembly.

Microsoft Intermediate Language (MSIL)

The code that is output by the C# compiler is written in a language called Microsoft Intermediate Language, or MSIL. MSIL is made up of a specific set of instructions that specify how your code should be executed. It contains instructions for operations such as variable initialization, calling object methods, and error handling, just to name a few. C# is not the only language in which source code changes into MSIL during the compilation process. All .NET-compatible languages, including Visual Basic .NET and Managed C++, produce MSIL when their source code is compiled. Because all of the .NET languages compile to the same MSIL instruction set, and because all of the .NET languages use the same runtime, code from different languages and different compilers can work together easily.

MSIL is not a specific instruction set for a physical CPU. It knows nothing about the CPU in your machine, and your machine knows nothing about MSIL. How, then, does your .NET code run at all, if your CPU can't read MSIL? The answer is that the MSIL code is turned into CPU-specific code when the code is run for the first time. This process is called "just-in-time" compilation, or JIT. The job of a JIT compiler is to translate your generic MSIL code into

machine code that can be executed by your CPU. You may be wondering about what seems like an extra step in the process. Why generate MSIL when a compiler could generate CPU-specific code directly? After all, compilers have always done this in the past.

There are a couple of reasons for this. First, MSIL enables your compiled code to be easily moved to different hardware. Suppose you've written some C# code and you'd like it to run on both your desktop and a handheld device. It's very likely that those two devices have different types of CPUs. If you only had a C# compiler that targeted a specific CPU, then you'd need two C# compilers: one that targeted your desktop CPU and another that targeted your handheld CPU. You'd have to compile your code twice, ensuring that you put the right code on the right device. With MSIL, you compile once. Installing the .NET Framework on your desktop machine includes a JIT compiler that translates your MSIL into CPU-specific code for your desktop.

Installing the .NET Framework on your handheld includes a JIT compiler that translates that same MSIL into CPU-specific code for your handheld. You now have a single MSIL code base that can run on any device that has a .NET JIT compiler. The JIT compiler on that device takes care of making your code run on the device.

Another reason for the compiler's use of MSIL is that the instruction set can be easily read by a verification process. Part of the job of the JIT compiler is to verify your code to ensure that it is as clean as possible. The verification process ensures that your code is accessing memory properly and that it is using the correct variable types when calling methods that expect a specific type. These checks ensure that your code doesn't execute any instructions that could make the code crash.

The MSIL instruction set was designed to make this verification process relatively straightforward. CPU-specific instruction sets are optimized for quick execution of the code, but they produce code that can be hard to read and, therefore, hard to verify. Having a C# compiler that directly outputs CPU-specific code can make code verification difficult or even impossible. Allowing the .NET Framework JIT compiler to verify your code ensures that your code accesses memory in a bug-free way and that variable types are properly used.

Metadata

The compilation process also outputs metadata, which is an important piece of the .NET codesharing story. Whether you use C# to build an end-user application or you use C# to build a class library to be used by someone else's application, you're going to want to make use of some already-compiled .NET code. That code may be supplied by Microsoft as a part of the .NET Framework, or it may be supplied by a user over the Internet. The key to using this external code is letting the C# compiler know what classes and variables are in the other code base so that it can match up the source code you write with the code found in the precompiled code base that you're working with.

Think of metadata as a "table of contents" for your compiled code. The C# compiler places metadata in the compiled code along with the generated MSIL. This metadata accurately describes all the classes you wrote and how they are structured. All of the classes' methods and variable information is fully described in the metadata, ready to be read by other applications. Visual Basic .NET, for example, may read the metadata for a .NET library to provide the IntelliSense capability of listing all of the methods available for a particular class. If you've ever worked with COM (Component Object Model), you may be familiar with *type libraries.* Type libraries aimed to provide similar "table of contents" functionality for COM objects.

However, type libraries suffered from some limitations, not the least of which was the fact that not all of the data relevant to the object was put into the type library. Metadata in .NET does not have this shortcoming. All of the information needed to describe a class in code is placed into the metadata. You can think of metadata as having all of the benefits of COM type libraries without the limitations.

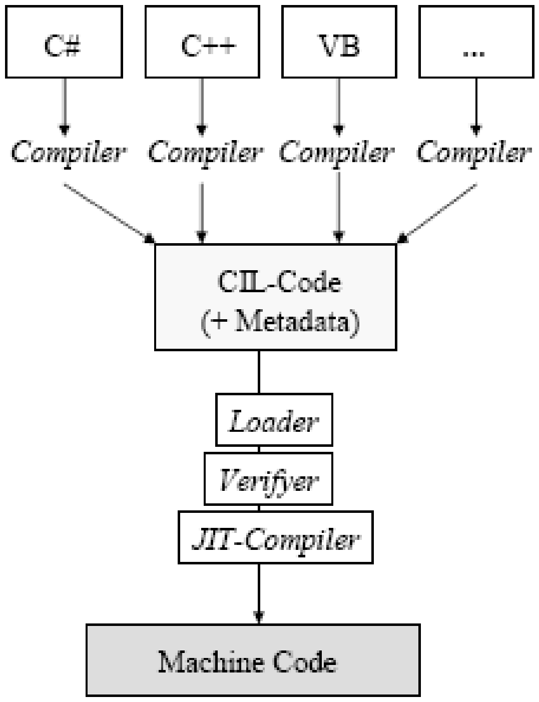
Assemblies

Sometimes, you will use C# to build an end-user application. These applications are packaged as executable files with an extension of .EXE. Windows has always worked with .EXE files as application programs, and C# fully supports building .EXE files. However, there may be times when you don't want to build an entire application. Instead, you may want to build a code library that can be used by others. You may also want to build some utility classes in C#, for example, and then hand the code off to a Visual Basic .NET developer, who will use your classes in a Visual Basic .NET application. In cases like this, you won't be building an application. Instead, you'll be building an *assembly*.

An assembly is a package of code and metadata. When you deploy a set of classes in an assembly, you are deploying the classes as a unit; and those classes share the same level of version control, security information, and activation requirements. Think of an assembly as a "logical DLL." If you're familiar with Microsoft Transaction Server or COM+, you can think of an assembly as the .NET equivalent of a package.

There are two types of assemblies: *private assemblies* and *global assemblies*. When you build your assembly, you don't need to specify whether you want to build a private or a global assembly. The difference is apparent when you deploy your assembly. With a private assembly, you make your code available to a single application. Your assembly is packaged as a DLL, and is installed into the same directory as the application using it. With a deployment of a private assembly, the only application that can use your code is the executable that lives in the same directory as your assembly.

If you want to share your code among many applications, you might want to consider deploying your code as a global assembly. Global assemblies can be used by any .NET application on the system, regardless of the directory in which it is installed. Microsoft ships assemblies as a part of the .NET Framework, and each of the Microsoft assemblies is installed as a global assembly. The .NET Framework contains a list of global assemblies in a facility called the *global assembly cache,* and the .NET Microsoft Framework SDK includes utilities to both install and remove assemblies from the global assembly cache.



SQL SERVER 2005

SQL Server 2005 is the successor to SQL Server 2000. It included native support for managing XML data, in addition to relational data. For this purpose, it defined an `xml` data type that could be used either as a data type in database columns or as literals in queries. XML columns can be associated with XSD schemas. XML data being stored is verified against the schema. XML is converted to an internal binary data type before being stored in the database.

Specialized indexing methods were made available for XML data. XML data is queried using XQuery. Common Language Runtime (CLR) integration was a main feature with this edition, enabling one to write SQL code as Managed Code by the CLR. SQL Server 2005 added some extensions to the T-SQL language to allow embedding XQuery queries in T-SQL. In addition, it also defines a new extension to XQuery, called XML DML that allows query-based modifications to XML data.

SQL Server 2005 also allows a database server to be exposed over web services using Tabular Data Stream (TDS) packets encapsulated within SOAP (protocol) requests. When the data is accessed over web services, results are returned as XML.For relational data, T-SQL has been augmented with error handling features (try/catch) and support for recursive queries with CTEs (Common Table Expressions). SQL Server 2005 has also been enhanced with new indexing algorithms, syntax and better error recovery systems.

Data pages are checksummed for better error resiliency, and optimistic concurrency support has been added for better performance. Permissions and access control have been made more granular and the query processor handles concurrent execution of queries in a more efficient way. Partitions on tables and indexes are supported natively, so scaling out a database onto a cluster is easier. SQL CLR was introduced with SQL Server 2005 to let it integrate with the .NET Framework.

SQL Server 2005 introduced "MARS" (Multiple Active Results Sets), a method of allowing usage of database connections for multiple purposes.SQL Server 2005 introduced DMVs (Dynamic Management Views), which are specialized views and functions that return server state information that can be used to monitor the health of a server instance, diagnose problems, and tune performance.

SQL SERVER CONFIGURATION MANAGER

SQL Server Configuration Manager is a new tool in SQL Server 2005. It is used to manage SQL Server 2005 services and connections. It has been developed as a Microsoft Management Console (MMC) plug-in application Its window is divided into a Console tree (left pane) and a Details pane. It can be managed through SQL Server services and connection configurations by navigating objects in the Console tree.

MANAGING SERVERS

- SQL Server
- SQL Server Agent
- SQL Server Browser
- SQL Server Integration Services
- SQL Server Anaylysis Services

SQL SERVER MANAGEMENT STUDIO

This tool is a new feature in SQL Server 2005. It replaces Enterprise Manager and Query Analyzer from earlier versions. It has been developed using a Visual Studio shell as a base. It follows the paradigm of Visual Studio, in which most tools are organized as tabbed, dockable, or floating windows. The registered server's pane allows viewing and managing parameters for connecting to servers. The tool includes both script editors and graphical tools which work with objects and features of the server.A central feature of SQL Server Management Studio is the

Object Explorer, which allows the user to browse, select, and act upon any of the objects within the server.

SQL SERVER 2005 FEATURES

- Database mirroring
- T-SQL (Transaction SQL) enhancements
- CLR integration
- Service Broker
- DDL triggers
- Ranking functions
- Row versioning-based isolation levels
- XML integration
- TRY...CATCH
- Database Mail
- **D**ata encryption

EASE OF INSTALLATION, DEPLOYMENT AND USE

SQL Server includes a set of administrative and development tools that improve ability to install, deploy, manage, and use SQL Server across several sites.

SCALABILITY

The same database engine can be used across platforms ranging from laptop computers running Microsoft Windows® 95/98 to large, multiprocessor servers running Microsoft Windows NT®, Enterprise Edition.

DATA WAREHOUSING

SQL Server includes tools for extracting and analyzing summary data for online analytical processing (OLAP). SQL Server also includes tools for visually designing databases and analyzing data using English-based questions. In this project SQL server is used because of the above features.

1.1 ORGANIZATION PROFILE

NR SOFTWARE SOLUTION

ABOUT US

NR software Solution started in the year 2015 with the clear objective to give solutions to all sectors throughout the World. NR is a software development and website design company in Coimbatore. NR is spearheaded by seasoned IT professionals and considers itself as one of the few website design companies in Coimbatore who are flexible enough to offer services ranging from website design to ecommerce solutions brochure design to corporate id packages, bespoke website designs to customized desktop applications. Nexus also offers offshore outsourcing and offshore staffing at highly competitive rates.

NR software Solution also provides world class services regarding web hosting and web space. We are offering optimum quality website hosting services to 100's of companies coming from all around India. As a reliable and experienced web host, we offer completely secure and cost-effective web hosting solutions. Backed by years of experience and having a large number of proficient professionals, NR believes in providing truly world class website hosting, domain hosting and domain name hosting services.

In addition to web page hosting and web hosting services, we also provide excellent domain registration hosting and dedicated web server hosting services. In our domain registration hosting services, we assist the customers in finding their domain name as per their preferences and requirements. Having a team of talented & experienced professionals and state of the art infrastructure, today NR software Solution stands at the forefront when it comes to providing world class server hosting services at affordable prices.

We are also providing the broadest range of software components and tools needed for enterprise applications. NR was founded with a vision in providing simplified solutions for complex global business and requirements. Based in Coimbatore we are the best IT Company working to design and deliver world class business solutions.

1.2 SYSTEM SPECIFICATION

1.2.1 HARDWARE CONFIGURATION

• System	:	i3
• Hard Disk	:	500 GB
• Monitor	:	14’ Colour Monitor
• Mouse	:	Optical Mouse
• Ram	:	4 GB

Software Requirements:

• Operating system	:	Windows 10
• Front End	:	Dotnet 2008
• Coding language	:	C#.net
• Data Base	:	Sql Server 2005

2.SYSTEM STUDY

2.1 EXISTING SYSTEM

- ❖ Independent of the development of cluster and high performance computing, the emergence of clouds and the MapReduce programming model has resulted in file systems such as the Hadoop Distributed File System (HDFS), Amazon S3 File System, and Cloud-Store. This, in turn, has accelerated the wide-spread use of distributed and parallel computation on large datasets in many organizations.
- ❖ Some of the earliest work in securing large-scale distributed file systems, for example, have already employed Kerberos for performing authentication and enforcing access control. Kerberos, being based on mostly symmetric key techniques in its early deployment, was generally believed to be more suitable for rather closed, well-connected distributed environments.
- ❖ On the other hand, data grids and file systems such as, OceanStore, LegionFS and FARSITE, make use of public key cryptographic techniques and public key infrastructure (PKI) to perform cross-domain user authentication.

2.1.1 DRAWBACKS

- ❖ The current design of NFS/pNFS focuses on *interoperability*, instead of efficiency and scalability, of various mechanisms to provide basic security. Moreover, key establishment between a client and multiple storage devices in pNFS are based on those for NFS, that is, they are not designed specifically for parallel communications. Hence, the metadata server is not only responsible for processing access requests to storage devices (by granting valid layouts to authenticated and authorized clients), but also required to generate all the corresponding session keys that the client needs to communicate securely with the storage devices to which it has been granted access.

- Consequently, the metadata server may become a performance bottleneck for the file system. Moreover, such protocol design leads to key escrow. Hence, in principle, the server can learn all information transmitted between a client and a storage device. This, in turn, makes the server an attractive target for attackers.
- Another drawback of the current approach is that past session keys can be exposed if a storage device's long-term key shared with the metadata server is compromised. We believe that this is a realistic threat since a large-scale file system may have thousands of geographically distributed storage devices. It may not be feasible to provide strong physical security and network protection for all the storage devices.

2.2 PROPOSED SYSTEM

- In this work, we investigate the problem of secure many to- many communications in large-scale network file systems that support parallel access to multiple storage devices. That is, we consider a communication model where there are a large number of clients (potentially hundreds or thousands) accessing multiple remote and distributed storage devices (which also may scale up to hundreds or thousands) in parallel.
- Particularly, we focus on how to exchange key materials and establish *parallel secure sessions* between the clients and the storage devices in the parallel Network File System (pNFS)—the current Internet standard—in an efficient and scalable manner. The development of pNFS is driven by Panasas, Netapp, Sun, EMC, IBM, and UMich/CITI, and thus it shares many common features and is compatible with many existing commercial/proprietary network file systems.
- Our primary goal in this work is to design efficient and secure authenticated key exchange protocols that meet specific requirements of pNFS.
- The main results of this paper are three new provably secure authenticated key exchange protocols. Our protocols, progressively designed to achieve each of the above properties, demonstrate the trade-offs between efficiency and security.
- We show that our protocols can reduce the workload of the metadata server by approximately half compared to the current Kerberos-based protocol, while achieving the desired security properties and keeping the computational overhead at the clients and the storage devices at a reasonably low level. We define an appropriate security model and prove that our protocols are secure in the model.

2.2.1 **FEATURES**

- The proposed system achieves the following three:

- ❖ Scalability – the metadata server facilitating access requests from a client to multiple storage devices should bear as little workload as possible such that the server will not become a performance bottleneck, but is capable of supporting a very large number of clients.
- ❖ Forward secrecy – the protocol should guarantee the security of past session keys when the long-term secret key of a client or a storage device is compromised.
- ❖ Escrow-free – the metadata server should not learn any information about any session key used by the client and the storage device, provided there is no collusion among them.

3.SYSTEM DESIGN AND DEVELOPMENT

3.1 FILE DESIGN

A file system provides the machinery to support the project tasks. At the highest level a file system is a way to organize, store, retrieve, and manage information on a permanent storage medium such as a disk. File systems manage permanent storage and form an integral part of all operating systems. There are many different approaches to the task of managing permanent storage. At one end of the spectrum are simple file systems that impose enough restrictions to inconvenience users and make using the file system difficult. In deciding what type of filing system is appropriate for a particular operating system, we must weigh the needs of the problem with the other constraints of the project. The two basic abstractions of files and directories form the basis of what a file system can operate on. There are many operations that a file system can perform on files and directories. All file systems must provide some basic level of support. Beyond the most basic file system primitives lay other features, extensions, and more sophisticated operations.

The Structure of a File is given the concept of a file, a file system may impose no structure on the file, or it may enforce a considerable amount of structure on the contents of the file. An unstructured, “raw” file, often referred to as a “stream of bytes,” literally has no structure. The file system simply records the size of the file and allows programs to read the bytes in any order or fashion that they desire. If a file system chooses to enforce a formal structure on files, it usually does so in the form of records. With the concept of records, a programmer specifies the size and format of the record, and then all I/O to that file must happen on record boundaries and be a multiple of the record length.

3.2 INPUT DESIGN

Input design is the process of converting user-originated inputs to a computer-based format. Input design is one of the most expensive phases of the operation of computerized system and is often the major problem of a system. Input design is a part of overall design, which requires careful attribute. Inaccurate input data are the most common cause of errors in data processing. The goal of designing input data is to make data entry as easy, logical and free from errors. In the system design phase input data are collected and organized into groups of similar data. In this project the input forms are the user request to access the cloud storage by means of providing their private and public key. Also the file selection and upload process are considers as the input.

3.3 OUTPUT DESIGN

Output design generally refers to the results and information that are generated by the system for many end-users; output is the main reason for developing the system and the basis on which they evaluate the usefulness of the application. Computer output is the most important and direct source of information to the user. Output design is very important phase because the output will be in an interactive manner. The key exchange between the user and the metadata server and the secured cloud data storage are the output in this project. The additional feature to the user can audit their stored file and get the result of the file corrupted information.

3.4 DATABASE DESIGN

The database design is a must for any application developed especially more for the data store projects. Since the chatting method involves storing the message in the table and produced to the sender and receiver, proper handling of the table is a must. In the project, login table is designed to be unique in accepting the username and the length of the username and password should be greater than zero. The different users view the data in different format according to the privileges given.

3.5 SYSTEM DEVELOPMENT

3.5.1 DESCRIPTION OF MODULES

- Meta Data Server
- Cloud Storage Devices
- Authenticated Key Exchange
- Data Auditing

META DATA SERVER

The entity that manages metadata is called a metadata server. pNFS separates the file system protocol processing into two parts: metadata processing and data processing. Metadata is information about a file system object, such as its name, location within the namespace, owner, permissions and other attributes. The meta data server generates a pair wise key for each cloud users and store their information authentication information. Also the user activities about the log details are monitored by the meta data server. The meta data server generates and provides a session key to the end user to access the cloud storage.

CLOUD STORAGE DEVICES

On the other hand, regular files data is striped and stored across storage devices or servers. Data striping occurs in at least a way: on a block-by-block basis. The data are splitted and as the fragments and those fragments are secured by the cryptography technique. The cryptography technique maintains the key inforamtion about the owner of a file. Unlike NFS, a read or write of data managed with pNFS is a direct operation between a client node and the storage system itself. Nevertheless, they can be extended straightforwardly to the multi-user setting, i.e., many-to-many communications between clients and storage devices.

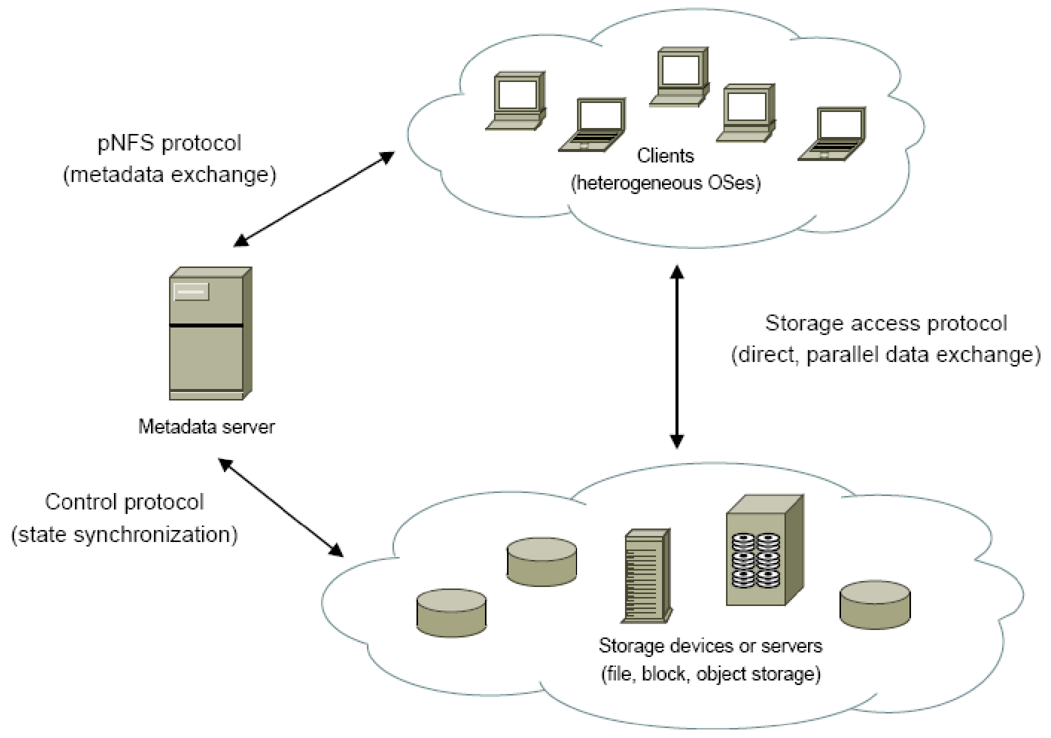
AUTENTICATED KEY EXCHANGE

We describe our design goals and give some intuition of a variety of pNFS authenticated key exchange (pNFS-AKE) protocols that we consider in this work. In these protocols, we focus on parallel session key establishment between a client and n different storage devices through a metadata server.Our primary goal in this work is to design efficient and secure authenticated key exchange protocols that meet specific requirements of pNFS. The session key is a temporary variable that provides a data store access for a specific time duration. While the end of a session key, the meta data server pass the information about the expiration of a session and the termination of cloud server usage.

DATA AUDITING

The end user can store and read the data which are stored in the cloud server. The stored data can auditable by them self using their pair wise keys. Also the data can be modified or deleted by the authorized cloud user.

ARCHITECTURE DIAGRAM



4.TESTING AND IMPLEMENTATION

The most important phase in system development life cycle is system testing. The number and nature of errors in a newly designed system depends on the system specifications and the time frame given for the design.

A newly designed system should have all the subsystems working together, but in reality each subsystems work independently. During this phase, all the subsystems are gathered into one pool and tested to determine whether it meets the user requirements.

Testing is done at two level -Testing of individual modules and testing the entire system. During the system testing, the system is used experimentally to ensure that the software will run according to the specifications and in the way the user expects. Each test case is designed with the intent of finding errors in the way the system will process it.

Testing plays a very critical role in determining the reliability and efficiency of software and hence is a very important stage in software development. Software testing is done at different levels. They are the unit testing and system testing which comprises of integration testing and acceptance testing.

TYPES OF TESTING

UNIT TESTING

This is the first level of testing. The different modules are tested against the specifications produced during the integration. This is done to test the internal logic of each module. Those resulting from the interaction between modules are initially avoided. The input received and output generated is also tested to see whether it falls in the expected range of values. Unit testing is performed from the bottom up, starting with the smallest and lowest modules and proceeding one at a time.The units in a system are the modules and routines that are assembled and integrated to perform a specific function. The programs are tested for correctness of logic applied and detection of errors in coding. Each of the modules was tested and errors are rectified. They were then found to function properly.

INTEGRATION TESTING

In integration testing, the tested modules are combined into sub-systems, which are then tested. The goal of integration testing to check whether the modules can be integrated properly emphasizing on the interfaces between modules. The different modules were linked together and integration testing done on them.

VALIDATION TESTING

The objective of the validation test is to tell the user about the validity and reliability of the system. It verifies whether the system operates as specified and the integrity of important data is maintained. User motivation is very important for the successful performance of the system.All the modules were tested individually using both test data and live data. After each module was ascertained that it was working correctly and it had been "integrated" with the system. Again the system was tested as a whole. We hold the system tested with different types of users. The System Design, Data Flow Diagrams, procedures etc. were well documented so that the system can be easily maintained and upgraded by any computer professional at a later.

SYSTEM TESTING

The integration of each module in the system is checked during this level of testing. The objective of system testing is to check if the software meets its requirements. System testing is done to uncover errors that were not found in earlier tests. This includes forced system failures and validation of total system as the user in the operational environment implements it. Under this testing, low volumes of transactions are generally based on live data. This volume is increased until the maximum level for each transactions type is reached. The total system is also tested for recovery after various major failures to ensure that no data are lost during the breakdown.

IMPLEMENTATION

Implementation is the process that actually yields the lowest-level system elements in the system hierarchy (system breakdown structure). The system elements are made, bought, or reused. Production involves the hardware fabrication processes of forming, removing, joining, and finishing; or the software realization processes of coding and testing; or the operational procedures development processes for operators' roles. If implementation involves a production process, a manufacturing system which uses the established technical and management processes may be required.

The purpose of the implementation process is to design and create (or fabricate) a system element conforming to that element’s design properties and/or requirements. The element is constructed employing appropriate technologies and industry practices. This process bridges the system definition processes and the integration process.

System Implementation is the stage in the project where the theoretical design is turned into a working system. The most critical stage is achieving a successful system and in giving confidence on the new system for the user that it will work efficiently and effectively. The existing system was a long process.

The proposed system was developed using Visual Studio .NET. The existing system caused a long time transmission process but the system developed now has a very good user-friendly tool, which has a menu-based interface, graphical interface for the end user. After coding and testing, the project is to be installed on the necessary system. The executable file is to be created and loaded in the system. Again the code is tested in the installed system. Installing the developed code in the system in the form of an executable file is implementation.

5.CONCLUSION

We proposed three authenticated key exchange protocols for parallel network file system (pNFS). Our protocols offer three appealing advantages over the existing Kerberos-based pNFS protocol. First, the metadata server executing our protocols has much lower workload than that of the Kerberos-based approach. Second, two our protocols provide forward secrecy: one is partially forward secure (with respect to multiple sessions within a time period), while the other is fully forward secure (with respect to a session). Third, we have designed a protocol which not only provides forward secrecy, but is also escrow-free.

BIBLIOGRAPHY

REFERENCE BOOKS

1. Bill Hamilton, "**Programming SQL Server 2005**", O'Reilly Media Publisher, 2006.
2. Elias M.Award,"System Analysis and Design", Galgotia Publications, Second Edition.
3. Daniel Solis, "Illustrated C# 2008", Apress Publisher, 2008.
4. David B. Makofske, Michael J. Donahoo, Kenneth L. Calvert, "TCP/IP Sockets in C#", Academic Press Publishers, 2004.
5. Richard Blum, "C# Network Programming", John Wiley & Sons Publishers, 2006.
6. Robin Dewson, "**Pro SQL Server 2005",** Apress Publisher.
7. Roger S. Pressman,"Software Engineering", Fourth Edition, 2005.

REFERENCE WEBSITES

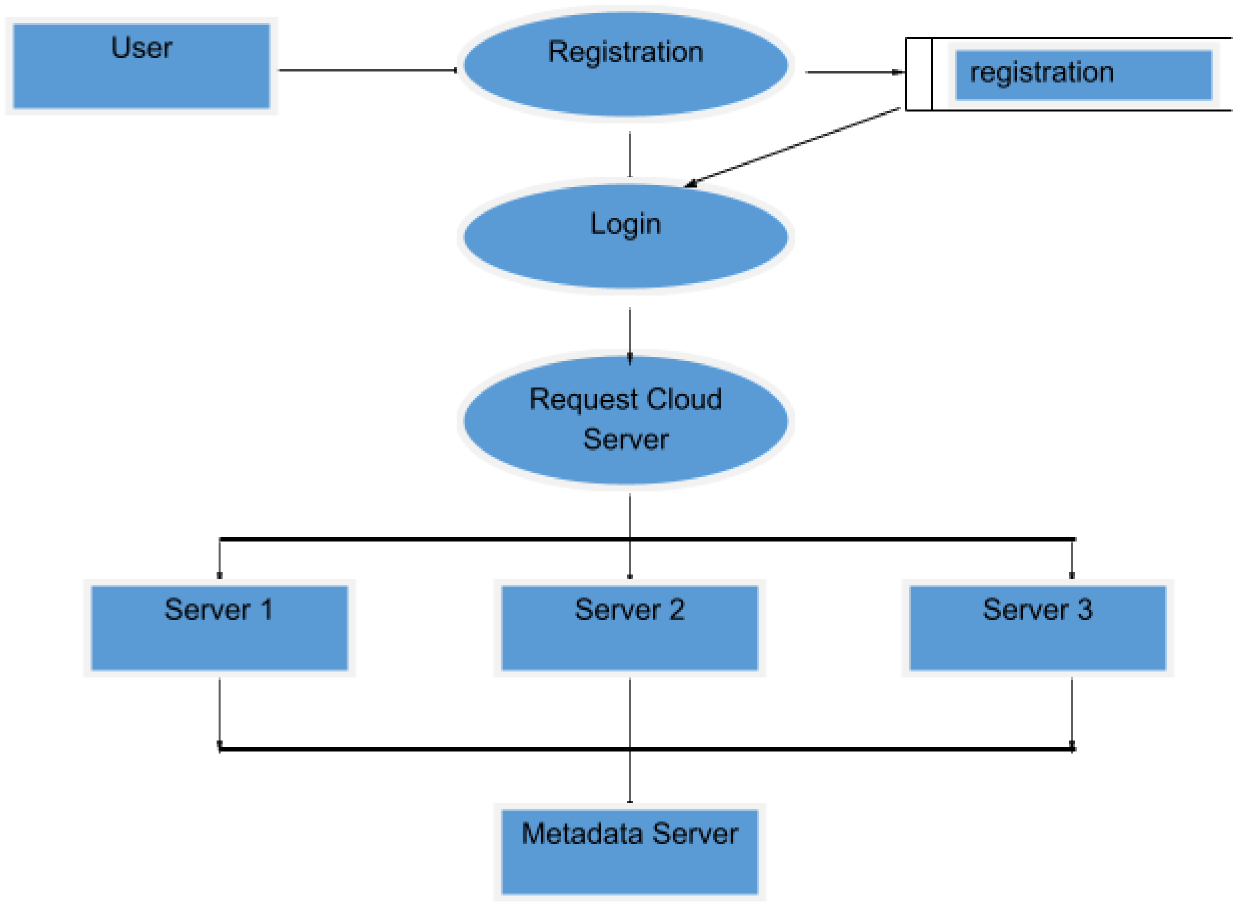
- www.dotnetspider.com
- www.programersheaven.com
- www.sql-server-performance.com
- www.developerfusion.com

- www.winsocketdotnetworkprogramming.com

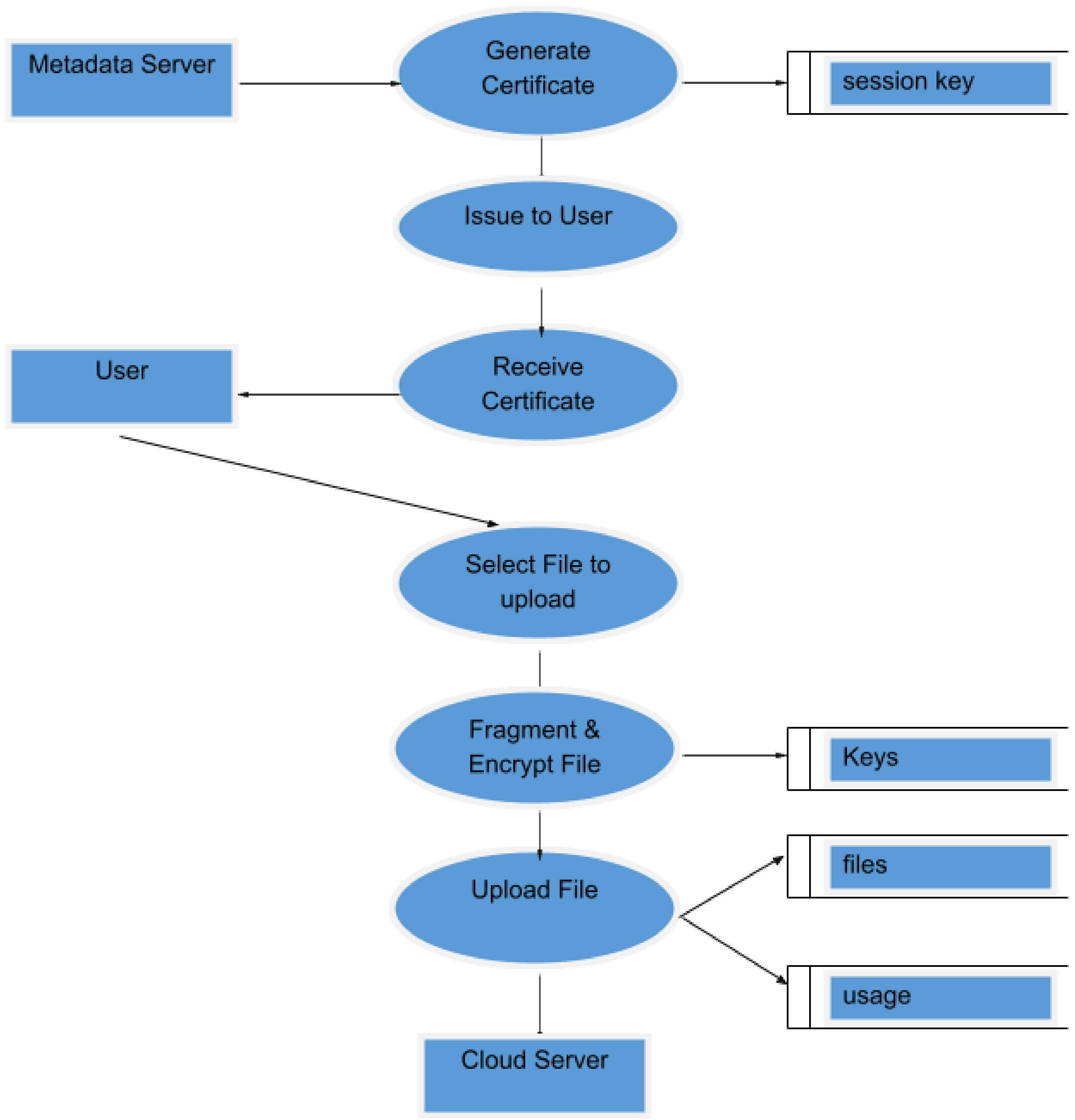
APPENDICES

A. DATA FLOW DIAGRAM

Level 0

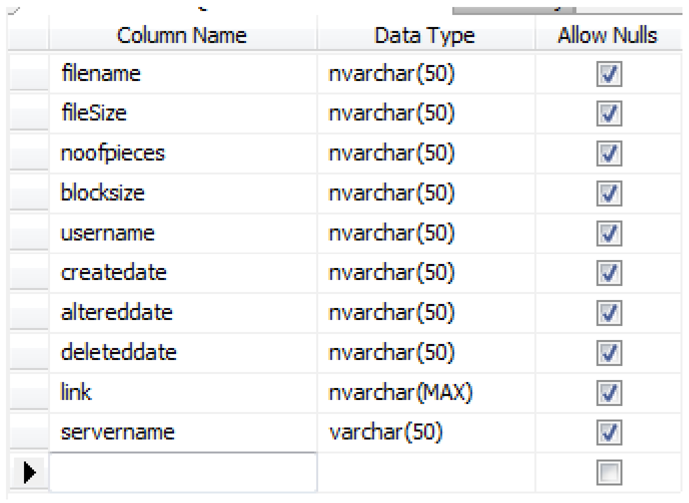


Level 1


Metadata Server
Generate Certificate
session key
Issue to User
Receive Certificate
User
Select File to upload
Fragment & Encrypt File
Keys
Upload File
files
usage
Cloud Server

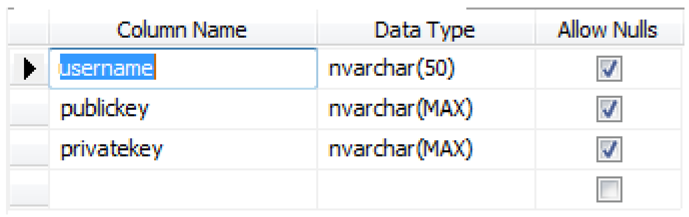
B. TABLE STRUCTURE

files



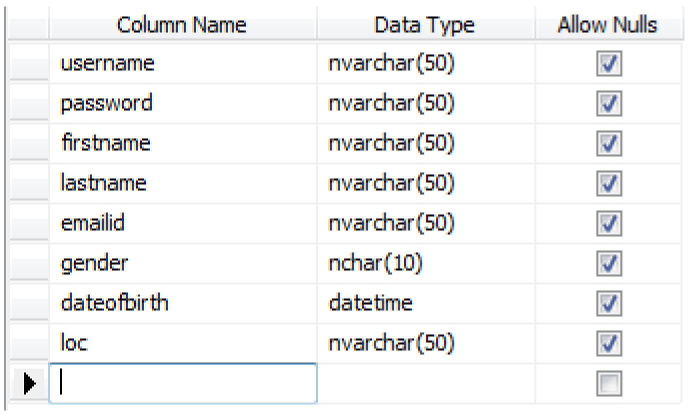
	Column Name	Data Type	Allow Nulls
	filename	nvarchar(50)	✓
	fileSize	nvarchar(50)	✓
	noofpieces	nvarchar(50)	✓
	blocksize	nvarchar(50)	✓
	username	nvarchar(50)	✓
	createdate	nvarchar(50)	✓
	altereddate	nvarchar(50)	✓
	deleteddate	nvarchar(50)	✓
	link	nvarchar(MAX)	✓
	servername	varchar(50)	✓
▶			☐

keys



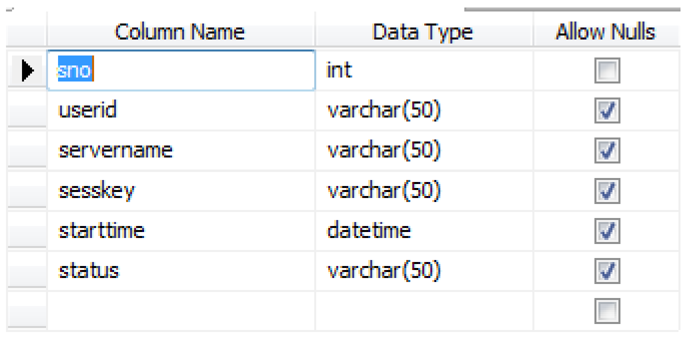
	Column Name	Data Type	Allow Nulls
▶	username	nvarchar(50)	✓
	publickey	nvarchar(MAX)	✓
	privatekey	nvarchar(MAX)	✓
			☐

registration



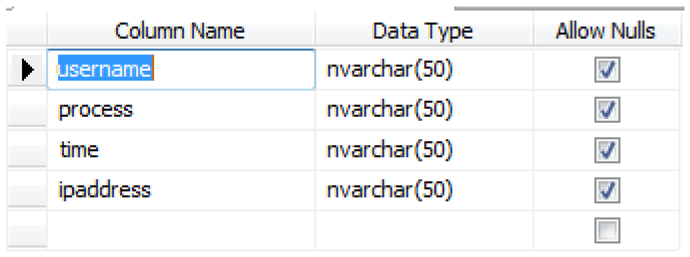
	Column Name	Data Type	Allow Nulls
	username	nvarchar(50)	✓
	password	nvarchar(50)	✓
	firstname	nvarchar(50)	✓
	lastname	nvarchar(50)	✓
	emailid	nvarchar(50)	✓
	gender	nchar(10)	✓
	dateofbirth	datetime	✓
	loc	nvarchar(50)	✓
▶			☐

session key



	Column Name	Data Type	Allow Nulls
▶	sno	int	☐
	userid	varchar(50)	✓
	servername	varchar(50)	✓
	sesskey	varchar(50)	✓
	starttime	datetime	✓
	status	varchar(50)	✓
			☐

usage



	Column Name	Data Type	Allow Nulls
▶	username	nvarchar(50)	✓
	process	nvarchar(50)	✓
	time	nvarchar(50)	✓
	ipaddress	nvarchar(50)	✓
			☐

C. CODING

Metadata Server

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
//using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace MetaDataServer
{
    public partial class login : Form
    {
        public login()
        {
            InitializeComponent();
        }

        private void textBox2_TextChanged(object sender, EventArgs e)
        {
        }

        private void button2_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        private void login_Load(object sender, EventArgs e)
        {

        }

        private void button1_Click(object sender, EventArgs e)
        {
            if (textBox1.Text == "admin" && textBox2.Text == "admin")
            {
                this.Hide();
                new admin().Show();
            }
        }
    }
}
```

Admin

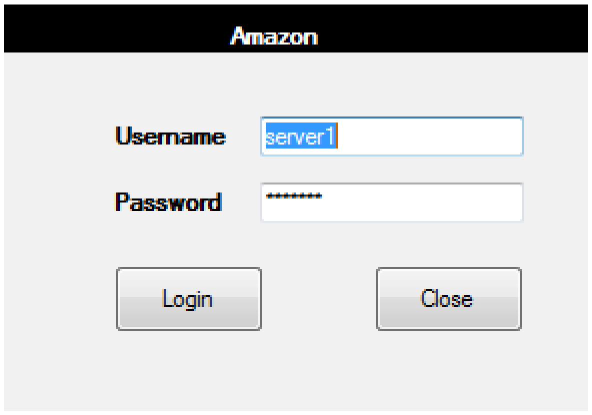
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
//using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Collections;
using ListNetworkComputers;
using System.IO;
using System.Data.SqlClient;
using remoteNamespace;
namespace MetaDataServer
{
    public partial class admin : Form
    {

        DialogResult dr;
        string tempcloud;
        SqlConnection con = new SqlConnection();
        SqlCommand cmd;
        SqlDataReader rd;
        string sqlserver;
        remoteListener server = new remoteListener();
        public admin()
        {
            InitializeComponent();
        }
        private void admin_Load(object sender, EventArgs e)
        {
            panel2.Visible = false;
            panel3.Visible = false;
            panel4.Visible = false;
            panel1.Visible = true;
            panel1.Dock = DockStyle.Fill;
                                              this.FormClosing     +=     new
    FormClosingEventHandler(this.admin_FormClosing);
        }
        private void admin_FormClosing(object sender, FormClosingEventArgs e)
        {
            server.stop();
            Application.Exit();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            panel4.Visible = false;
            panel3.Visible = false;
            panel1.Visible = false;
            panel2.Visible = true;
            panel2.Dock = DockStyle.Fill;
        }
        private void button10_Click(object sender, EventArgs e)
        {
            if (textBox2.Text != "")
            {
                sqlserver = textBox2.Text;
                panel4.Visible = false;
                panel3.Visible = false;
                panel2.Visible = false;
                panel1.Visible = true;
                                        con.ConnectionString = "server=" + sqlserver +
    ";database=server;integrated security=true;";
                button1.Enabled = false;
                timer1.Enabled = true;
                timer1.Start();
                server.Start(tempcloud, sqlserver);
            }
            else
            {
                MessageBox.Show("Enter All The Values");
            }
        }
```

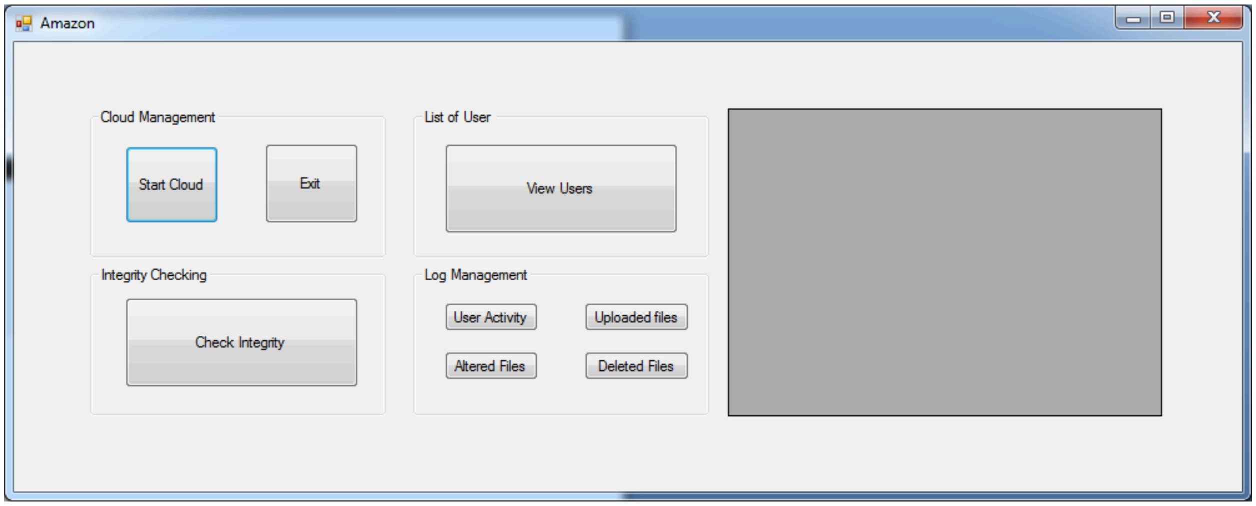
```
private void button17_Click(object sender, EventArgs e)
{
    panel4.Visible = false;
    panel3.Visible = false;
    panel2.Visible = false;
    panel1.Visible = true;
}

private void button2_Click(object sender, EventArgs e)
{
    server.pause();
    button1.Enabled = true;
   // button12.Enabled = false;
   // button13.Enabled = true;
}



private void button4_Click(object sender, EventArgs e)
{
    server.stop();
    Application.Exit();
}

private void button8_Click(object sender, EventArgs e)
{

}
```

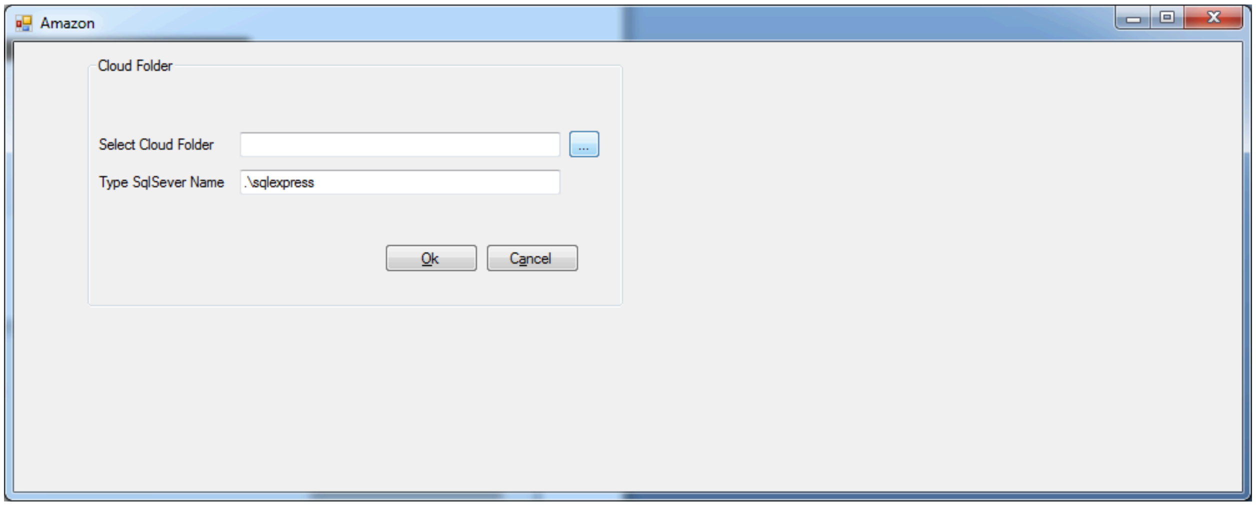
```
private void button18_Click(object sender, EventArgs e)
{
    panel1.Visible = true;
    panel2.Visible = false;
    panel3.Visible = false;
    panel4.Visible = false;

}


private void button7_Click(object sender, EventArgs e)
{

    try
    {
        panel1.Visible = false;
        panel2.Visible = false;
        panel4.Visible = false;
        panel3.Visible = true;
        panel3.Dock = DockStyle.Fill;


        if (sqlserver != "")
        {
```

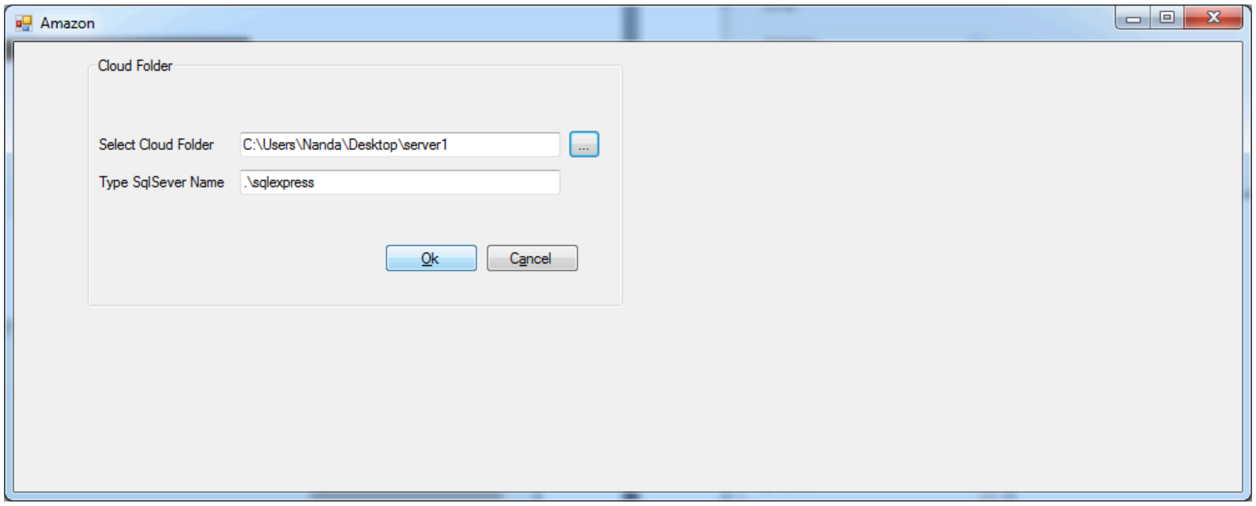
```
con.Open();
cmd = new SqlCommand("select * from [registration]", con);
DataSet ds = new DataSet();
DataTable dt = new DataTable();
dt.Columns.Add("UserName");
dt.Columns.Add("Password");
dt.Columns.Add("FirstName");
dt.Columns.Add("LastName");
dt.Columns.Add("Email-ID");
dt.Columns.Add("Gender");
dt.Columns.Add("Dob");
dt.Columns.Add("Location");
rd = cmd.ExecuteReader();
while (rd.Read())
{
    DataRow dr = dt.NewRow();
    dr["UserName"] = rd[0].ToString();
```

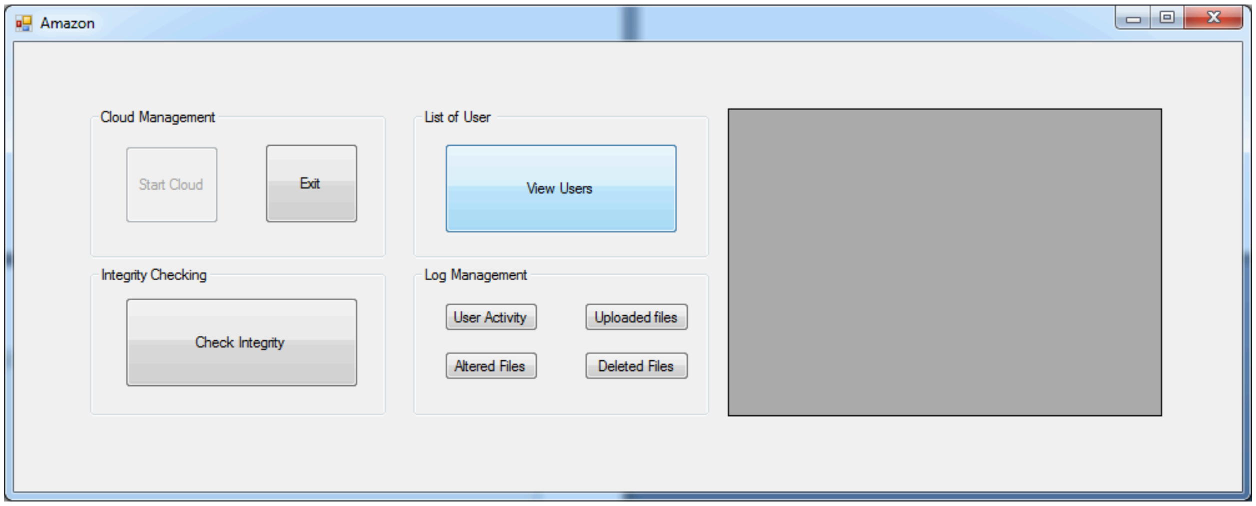
D. SAMPLE INPUT

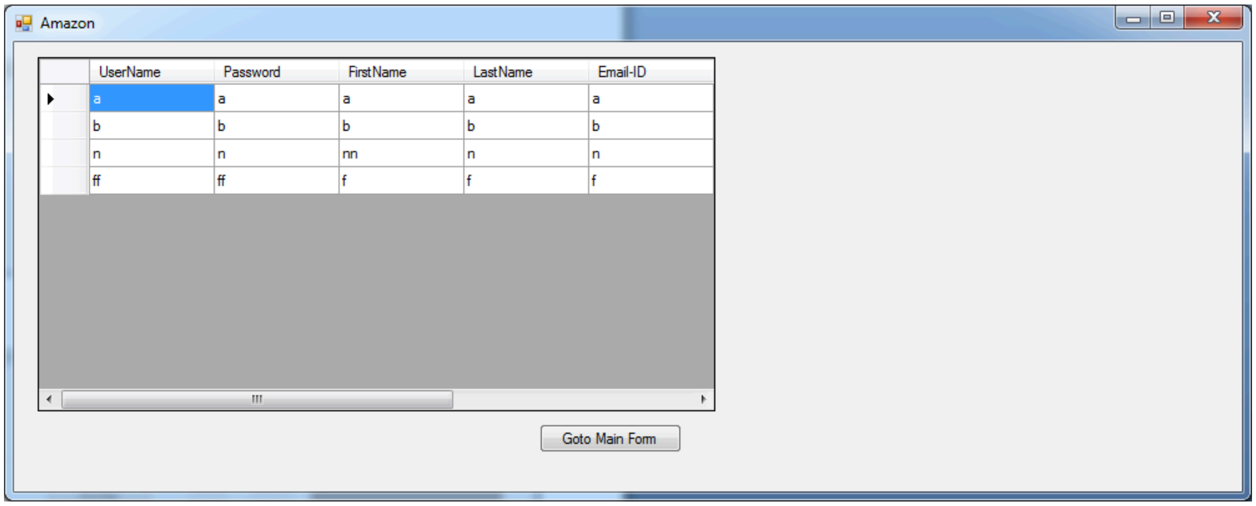




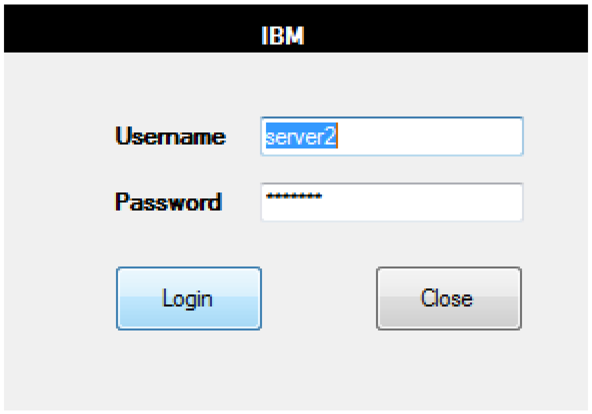

Amazon
Cloud Folder
Select Cloud Folder
...
Type SqlSever Name
.\sqlexpress
Ok
Cancel


Amazon
Cloud Folder
Select Cloud Folder
C:\Users\Nanda\Desktop\server1
...
Type SqlSever Name
.\sqlexpress
Ok
Cancel

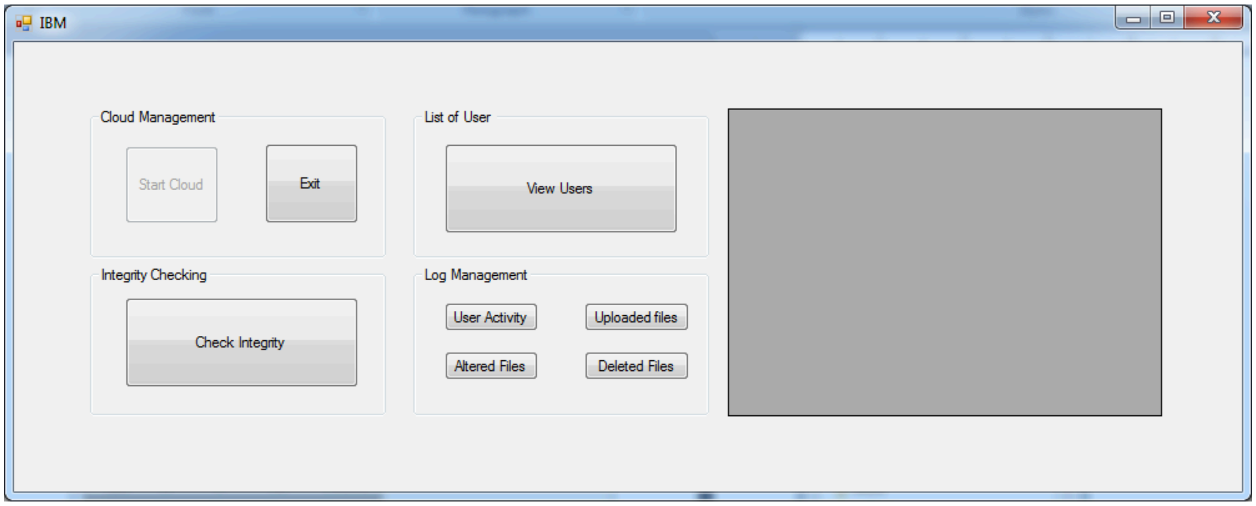


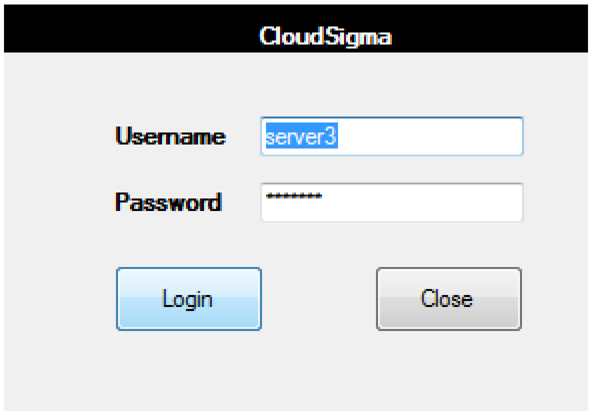


UserName	Password	FirstName	LastName	Email-ID
a	a	a	a	a
b	b	b	b	b
n	n	nn	n	n
ff	ff	f	f	f

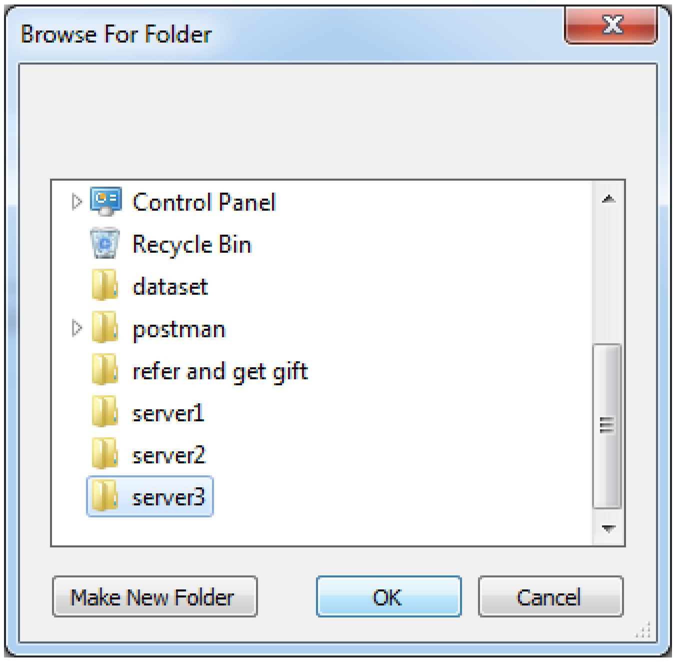


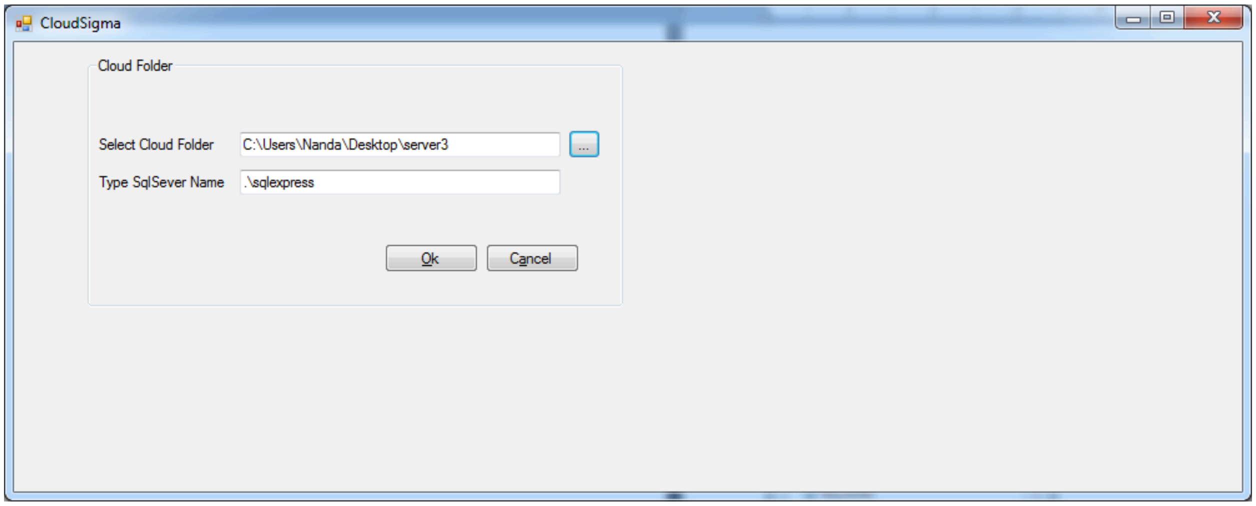
Username: server2
Password: ********





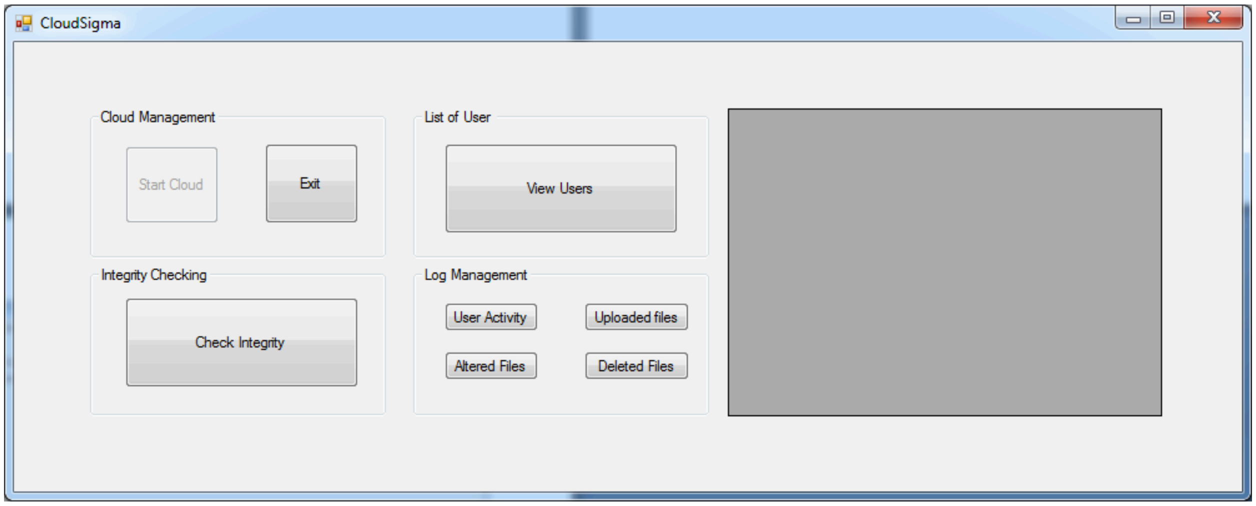
Username: server3
Password: ********

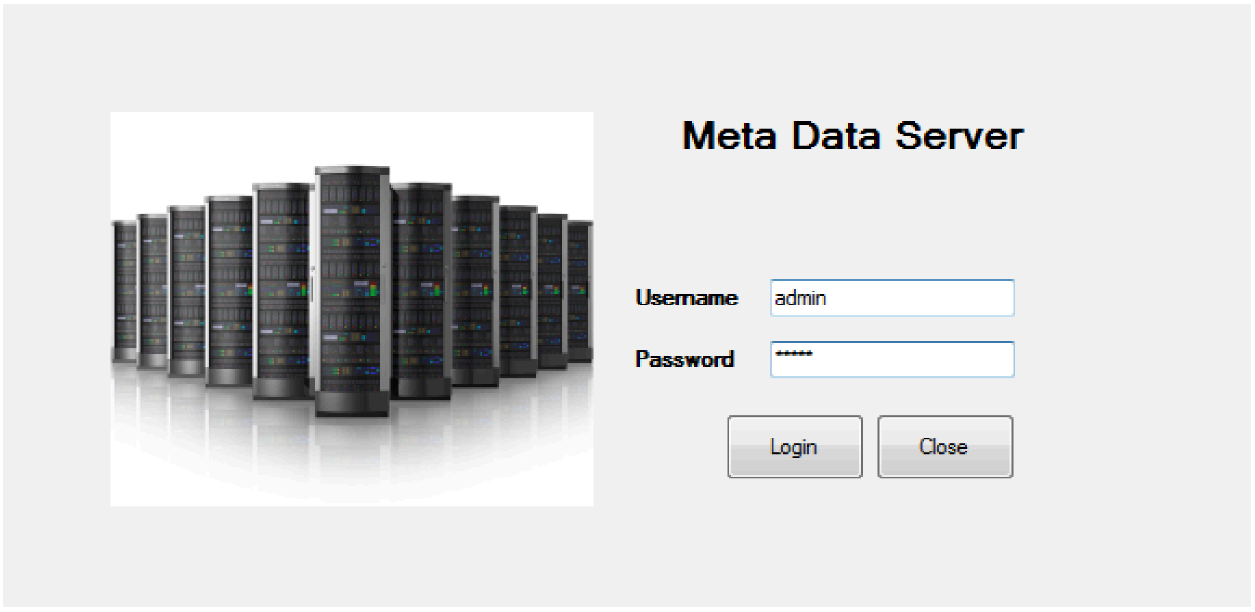


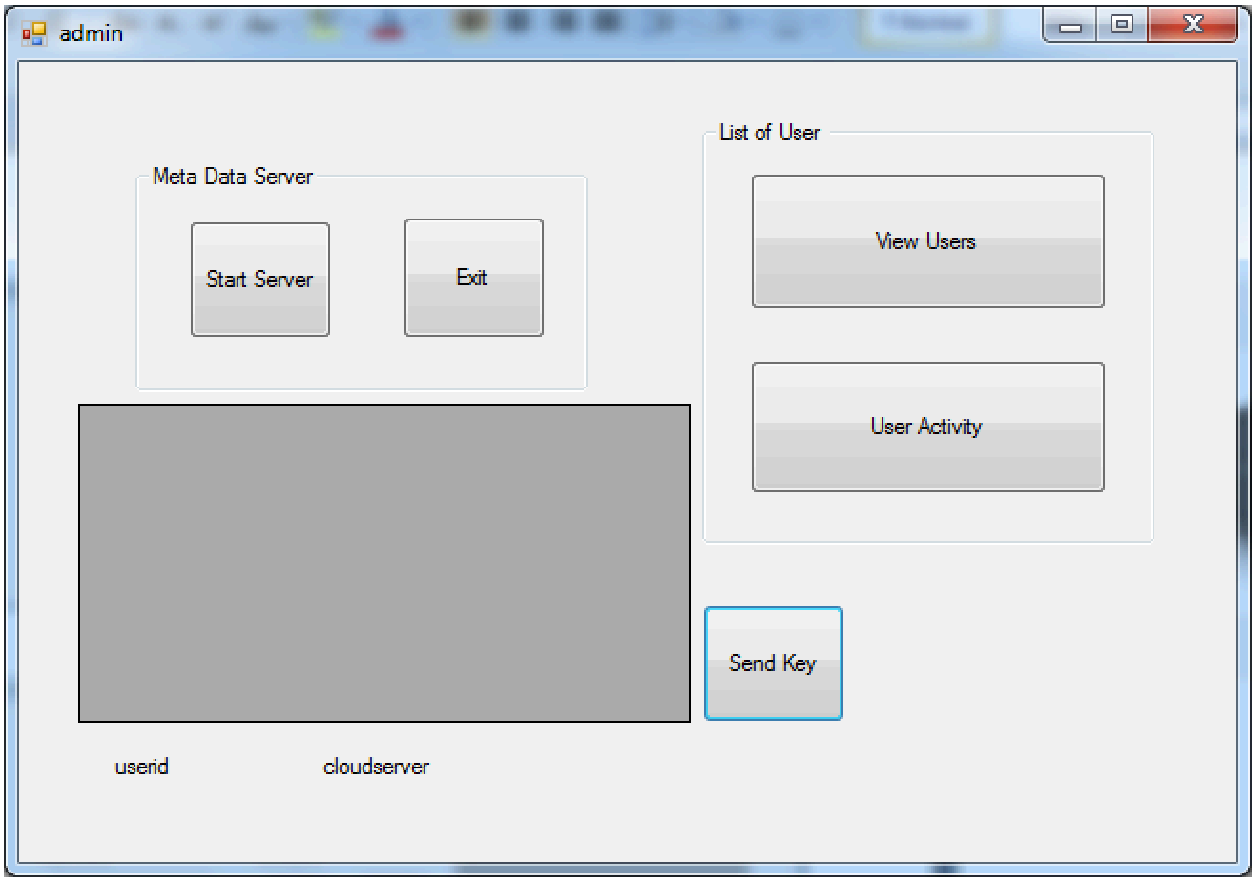


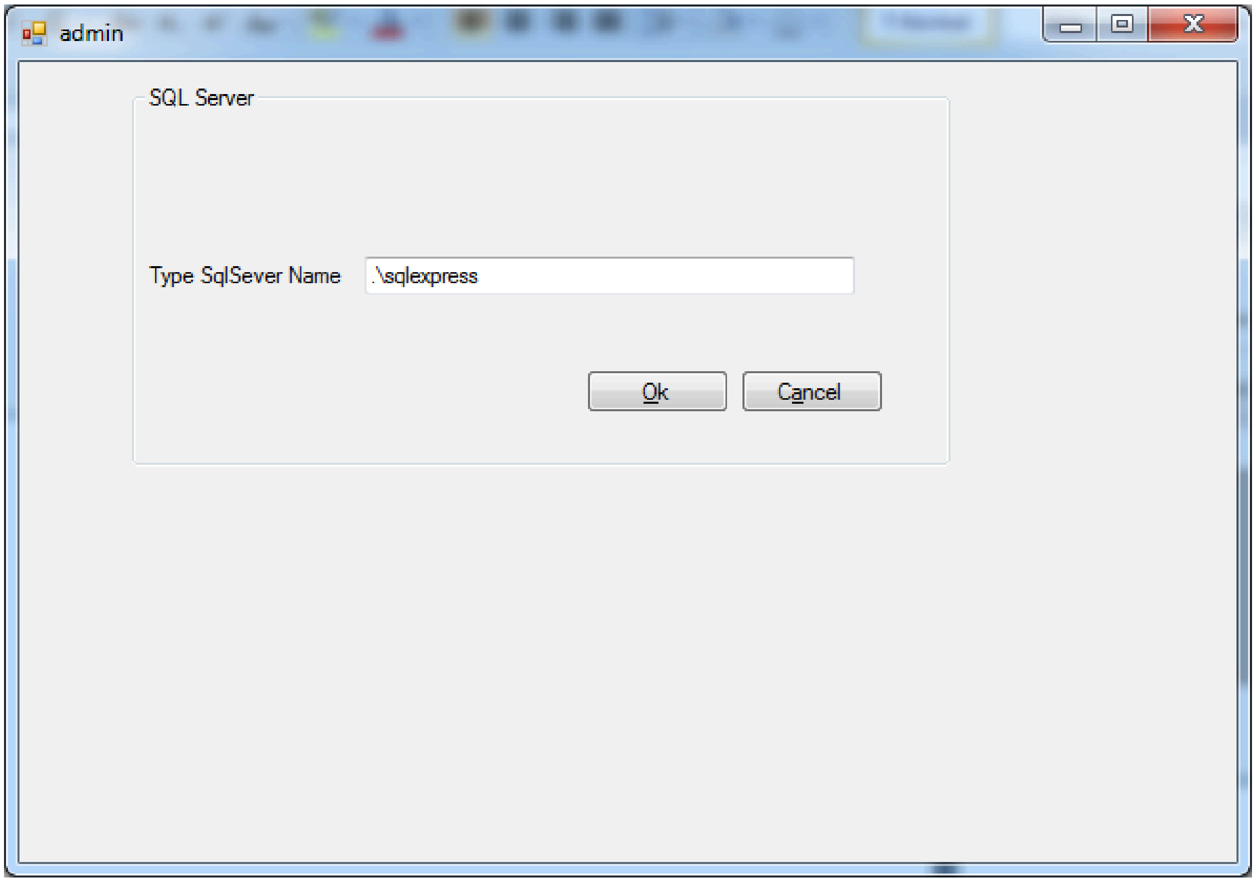
Select Cloud Folder: C:\Users\Nanda\Desktop\server3

Type SqlSever Name: .\sqlexpress




Meta Data Server
Username
admin
Password
******
Login
Close

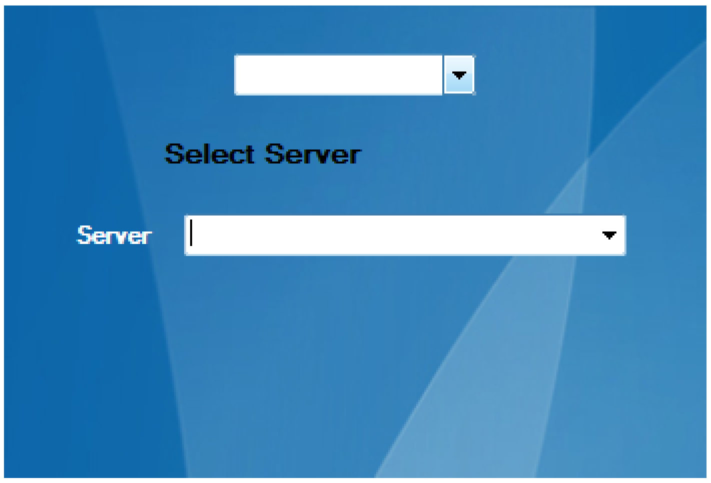

admin
Meta Data Server
Start Server
Exit
List of User
View Users
User Activity
Send Key
userid
cloudserver


admin

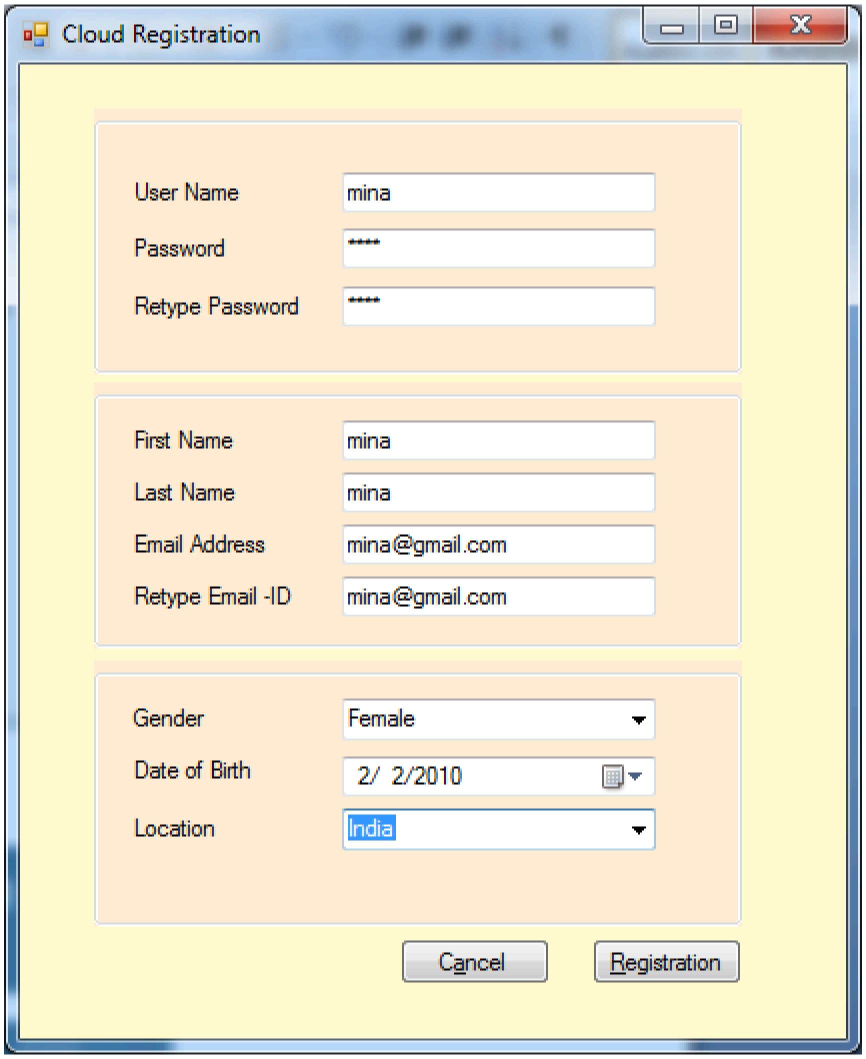
SQL Server

Type SqlSever Name: .\sqlexpress

client


Select Server

Server:


Cloud Registration

User Name	mina
Password	****
Retype Password	****
First Name	mina
Last Name	mina
Email Address	mina@gmail.com
Retype Email -ID	mina@gmail.com
Gender	Female
Date of Birth	2/ 2/2010
Location	India