High-Level Description of your Software
Instant Messenger / Chat with Strong Encryption - Java

Instant Messaging has quickly become a ubiquitous part of our everyday computing routine. We use it constantly to communicate with co-workers and family or friends. Unfortunately there are very few messaging clients which support strong encryption to guarantee the security of your messages, and even fewer that can do it across all platforms.

For our project we have decided to address these concerns and write and instant messaging application that is based on the principle of Public Key Encryption. Every message that is sent between the client and the server will be encrypted using the appropriate public key. We can guarantee the identity of the sender and guarantee the integrity of the passed message.

Two additional features we chose to support to really distinguish this application were the use of skins and pluggable applications. Skins allow the look and feel of the program to change to suit the environment you are working in – or make you feel like you are in a different environment all together!

The pluggable application comes in handy because it can be extended to any communication need. It is possible to have share web pages, play games or even video conference using this technology and guarantee that every part of that conversation is private and secure.

The entire project is turn key, there is no software to install or configure, just a simple self-contained application that automatically detects and configures everything the end-user needs.

System Components
Java formed the foundation for all the work on this project. We relied heavily on the built in classes that it supplied, particularly the networking (sockets), swing (GUI) and BigInteger (Cryptography) classes to form the core of the software. Our software has been successfully run on several Windows platforms, Solaris and Linux. The only requirement the computer has the JRE or JDK 1.2 or later (preferably 1.3 or 1.4). Since public key cryptography is quite math intensive, a faster processor makes quite a difference, but anything in the 300Mhz range or faster should be sufficient.

An external class we used to provide the skin effect is the Skin Look And Feel components which can be found at their homepage: http://www.l2fprod.com. This package allows the use of virtually any Qt or Gtk theme (KDE or GNOME) to set the look and feel for the application.
Coolness Factor
Several things really set this instant messaging application apart from the rest. First is that cryptography is always enabled, so you never have to worry about the security of your conversation. The second is the skins. This is the first and only instant messenger to fully support both Qt and Gtk themes. The third is it is written entirely in Java which makes it totally portable without even recompiling the code. Included in one single JAR file is everything the program needs to run. No need to install, update or manage anything.

System Capabilities
- Pass text messages back and forth, instant messaging
- Send files back and forth
- Pluggable applications can use the same framework to exchange data
- All traffic is encrypted so the end-user is 100% worry free
- Skinnable interface provides visual enhancements or the illusion of a different platform
- Platform independence means users can easily communicate across any computer

Individual Contributions
The project was divided mostly into three parts, the encryption engine, the server and the GUI. Tom handled all the server programming, Megan did the GUI and Greg did the encryption engine. A significant part of the programming was done side-by-side and through pair programming. This allowed for immediate feedback on problems or questions regarding implementation issues. Greg also wrote the pluggable interface and the Tic-Tac-Toe game.

Accomplishments
We all felt very good about what we learned and accomplished. There were so many aspects of GUI and Network programming that we might have brushed upon but never actually used it was very beneficial for all of us.

One thing we would like to have done would have been to use the JDBC and a database to store user information instead of just inside java and text files. It was mostly due to lack of planning and available resources.

Lessons Learned: Working as a Team
This project would not have been possible on a individual basis, due partly to time constraints but mostly to talent. Not one of us had the skills we needed to complete the project completely without the help and skills of other team members.

Our group meetings were very beneficial. We tried to keep them short and focus on the problems we were having and planning for the future. A little more planning during the meetings would have been beneficial long term, and the Olympics certainly threw us off.
Lessons Learned: Building a Large Software System
Start early and plan well. There was a lot more coding that took place as the due date for the software came closer. We could have made our lives a lot more manageable the last week if we had spent some more time before. Constancy would have been useful to as some weeks we put in 20 or more hours and other weeks it was just 1 or 2. If we could have averaged that out to more like 5 solid hours a week guaranteed, the package could have been more complete and polished. We probably would have had a lot more fun working on it as well.

It was very interesting to be contacted by Morgan Stanley Dean Whitter to actually put our software in use for their secure communications. We considered the enterprise when we began designing the software but failed to see some of their basic needs. It may not have changed our end product, but we could have been ready with responses and been more helpful to them.

Lessons Learned: Time Machine
Design: I think we would have stated our needs better and looked at the available resources. One unforeseen problem was the release of the Java 2 1.4 which added a lot of functionality that would have been beneficial for us, but due to issues with the lab we could not always rely on having up to date tools. As part of our design was built around using 1.4, we had to make a few changes down the road and come up with work around solutions for some of the problems that caused.

More on the design would have been to explicitly lay out our entire messaging protocol better. It went through far to many revisions which made it hard to keep our code synchronized and bug free. There were changes all the way up to the last week which could have been taken care of long before that. It probably would have allowed us to work more efficiently during the time we set aside as well.

Implementation: We would really have liked to use the JDK 1.4, especially for the enhancements in the security packages, network tools and thread management. It would seem Sun finally got some implementation details right for those things in version 1.4. Since we were basically forced to use version 1.3 in the lab (and for the demo) we had to rely on other methods to get done what we needed to. We also used JBuilder (which changed from version 5 to 6 in January), Forte and NetBeans to do our Java development which created some strain on maintaining source compliance, but nothing too serious.

Management: It would have been nice to use CVS or some other versioning software instead of structured FTP site. We each had a directory and basically announced through email the changes we made and where to find the latest code. Some sort of versioning software would have made it much simpler to make sure you had the latest code and you were not stomping all over what someone else was doing.