Security Systems for Distributed Models in Ptolemy II

Rakesh Reddy
Carnegie Mellon University
rnreddy@andrew.cmu.edu

Abstract

Ptolemy II is a powerful software package that models heterogeneous systems. Its infrastructure offers the ability to create distributed models, but lacks the security mechanisms to protect data being transferred in these models. The lack of security can lead to many potential threats and breaches of information. Potential attackers can jeopardize system stability by manipulating data in transit or by transmitting harmful models. Distributed models may also pass security sensitive data where outside parties should not be able to determine the information being passed. The current infrastructure however has no means to keep such information secure from eavesdropping. Eliminating these security threats requires the implementation of various cryptographic services including ciphers and digital signatures. The Sun Java Cryptography Architecture and Java Cryptography Extension packages provide the tools needed in eliminating these breaches of security. The semantics used by the above-mentioned Java packages allow for the creation of actors in which a diverse number of algorithms can be used with minimal knowledge of cryptography. By creating Ptolemy II actors that implement cryptographic services, we achieve a flexible yet, powerful defense of potential security leaks in distributed models. The result is being able to send and receive data Ptolemy II distributed models without the concern of information leaks or receiving adverse applications.

1. Introduction

As technology grows so does the complexity of problems we seek to solve. The complexity of these solutions often calls for the use of computers. However, even with the development of faster machines, more and more problems arise in which it becomes impractical for a single computer to complete the necessary task. The logical solution is to use more than one system. The result of this idea is distributed models.

Distributed computing, a form of distributed modeling, relies on the fact that multiple systems can process information much quicker than a single system. This is an excellent mechanism for solving computationally intensive problems that require large amounts of data to be processed. These models involve sending chunks of data to multiple systems in order to solve the given problem. A wide spread example of distributed computing is the Search for Extraterrestrial Intelligence at home (SETI@home). The basis of this project is to analyze data from space to determine if intelligent life exists outside of earth. The SETI@home software sends small pieces of data to be analyzed by home systems while they are not being used. The results of the analysis are then sent back to the SETI@home server. The use of distributed computing allows for significant amounts of data to be processed that would otherwise be impractical. The processing of information at this distributed level has led to many more data intensive applications, including the analysis of protein folding and determining the security of cryptographic algorithms.

In addition, distributed models are not limited to computing vast amounts of data. Several control applications exist where it is physically impossible to control a system with only one computer, creating a need for distributed control models. One such scenario is the Partners for Advanced Transit and Highways (PATH) project. Part of the project is to allow cars to communicate with each other in order to avoid collisions or improve the flow of traffic. It is physically impossible for a single computer system to do this for several reasons, most notably that a single system would have to tell millions of cars how to operate. The difficulty in communicating with so many cars simultaneously in itself is a seemingly insurmountable task using one computer. The answer lies in cars wirelessly transmitting information to each other and independently processing information to detect potential accidents rather than relying on a single system.

Although distributed computing is an excellent solution to the aforementioned problems, the issue of security must be addressed. With the large amount of information being passed around, many potential threats could jeopardize distributed systems in a variety of ways. Information can be transmitted by a malicious source that is potentially harmful to components of a distributed control system. For example someone could

use the PATH system to cause cars to crash. The lack of security would also make it simple to intercept data that is security sensitive.

2. Background Information

2.1. Ptolemy II

Ptolemy II is a Java based software package that is used to model heterogeneous systems using models of computation known as actors. The infrastructure of Ptolemy II allows for the distributed modeling of systems including distributed data processing and distributed controls. A key feature of distributed modeling within Ptolemy II is the use of mobile models. Mobile models are models that can be passed through distributed computing to perform operations as specified by the model. Mobile models go beyond the simple transmission of data that many distributed models are limited to by adding functionality to already existing models on a system. These models alter how the main model operates during runtime. This capability vastly expands the functionality of distributed computing that could not be achieved with the infrastructure of standard distributed systems.

The added functionality arising from mobile models also creates more security threats. Ptolemy II's security mechanism is lacking. The only form of security that exists is the Java *Sandbox*. It is a part of the Java Virtual Machine (JVM) that protects system from potential attack by the applications it runs. There are many limitations to the sandbox. JVM classes are placed in a dichotomy of either being trusted or not trusted. There is no form of limited trust where an application could access certain data on a system or write some information to the system. Furthermore if any part of an application is not trusted then the whole application can not be run. This dichotomy renders a result that either sacrifices system security or hinders the functionality of a distributed system. The sandbox provides no mechanism to protect data being transported either. The solution to the security dilemma is to implement various cryptographic services that ensure the protection of information and the ability to trust such information.

2.2 Ciphers

Ciphers are mechanisms used to encrypt or decrypt data. Ciphers are based on a key in conjunction with an algorithm that is used to transform data. Keys are pieces of data that are required to perform a cipher operation (decryption or encryption) on a piece of data. The purpose of a cipher is to encrypt a message so it is kept secret from anyone who does not have the necessary key. The original data transformed by the key can be transformed back by using a key. The key used depends on whether the cipher is symmetric and asymmetric.

2.2.1. Symmetric Cipher. Symmetric cryptography, also known as "secret key" or "private key" cryptography, is based on keeping the knowledge of the key private. A simple example would be shifting every letter by 3 and wrapping the last letters around. In this case the key would be 3 and would need to be kept secret. With the current computational power, some symmetric ciphers can be broken in a brute force manner using heuristics and other information such as the tendency for certain character to repeat. This has led to the development of more complex algorithms that computationally impractical to decode at this time.

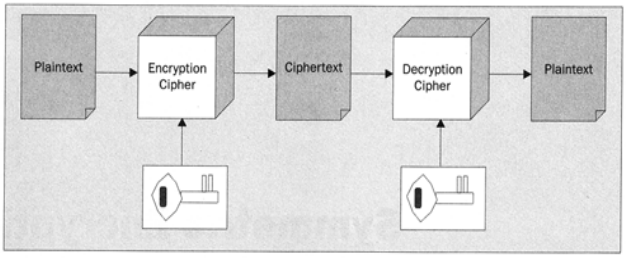

Figure 1: Symmetric Cipher process [4].

There are several other important features that become when using symmetric algorithms. Encrypting data in chunks as opposed to single characters makes it much more difficult to decrypt the data. Algorithms that offer sufficient security use block ciphering. Block ciphering requires two more components: padding and mode. The padding scheme determines what data will be used to fill in a partially filled block during encryption. The mode determines how the cipher should be applied. One type of mode is the Electronic Code Book which is similar to a replacement scheme. Another scheme is Cipher Block Chaining which uses the prior block to encrypt the next block.

The main weakness of symmetric cryptography lies in transporting the key since sending it with the message would allow anyone to decrypt it. Another threat to symmetric cryptography, as mentioned before, is a brute force attack which takes data and tries to manipulate it to get the original data.

2.2.2. Asymmetric. Unlike symmetric ciphers, asymmetric ciphers use key pairs. These key pairs comprise of a public key and a private key. The public key can be publicly known while only one person knows the corresponding private key. The message is encrypted using a public key and is sent to the person

with the corresponding private key. The private key is then used to decrypt the message. This is made possible by the fact that if, a message is encrypted using one key, it can only be decrypted with the other key of the pair. This key system eliminates the issues of needing to keep a key secret during transfer.

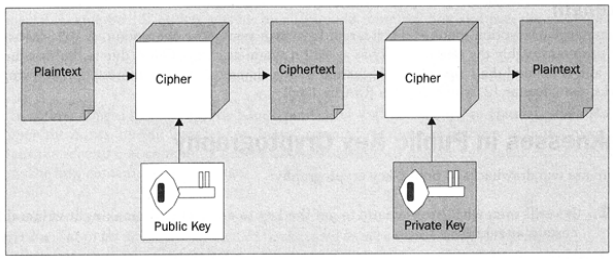

Figure 2: Asymmetric Cipher process [4].

The ease of transporting keys for asymmetric ciphers would lead one to believe that symmetric algorithms would be unnecessary. There are several reasons as to why this is not true. The main issue is asymmetric algorithms are computationally demanding compared to symmetric algorithms. Using asymmetric algorithms to encrypt large amount of data can be up to a thousand times slower than using symmetric algorithms [4]. Asymmetric ciphers can also suffer from man-in-the-middle attacks in which the key is tampered with. Another issue that becomes important with asymmetric ciphers is the random number generator used to make keys. This becomes a difficult task since randomness is something that eludes the mechanics of computers. Using weak PRNG (pseudo random number generators) provides an easier method to determine the keys in algorithms such RSA.

2.2.3. Hybrid Cryptography. In response to the drawbacks of symmetric and asymmetric ciphers, it is essential to combine the two. This is known as hybrid cryptography. Hybrid cryptography is achieved by encrypting a file using a symmetric cipher. The private key of the symmetric cipher is then encrypted using an asymmetric algorithm; this model addresses the main weaknesses of each cipher. In the case of symmetric ciphers, the issue of transferring the private key is eliminated for the key is encrypted using a public key cipher in which the public key can be known. The problem of asymmetric ciphers being slow for encryption is eliminated by merely encrypting the secret key.

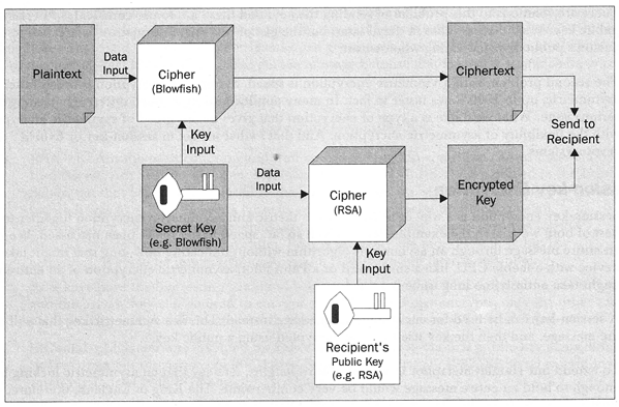

Figure 3: Hybrid Encryption process [4].

2.3. Message Digests

Message Digests are a method to determine that a message was not altered while in transit. The basis of a message digest is a hash function that has several key properties. The input length can be of any length while the output is some fixed length. The hash function should also be one way meaning that the original message can not be determined from the message hash. The hash function in message digests should also be collision free. Given a message x with a hash, H(x), it should be computationally infeasible to find a message y where H(y) = H(x). If this weren't true then one could easily modify a message and manipulate it so the same hash result is produced. This would make the point of a message digest useless.

One of the more recent message digests algorithms that is in wide use is the Secure Hash Algorithm (SHA); MD5 is another common hash that is widely used. SHA is slightly slower but more secure because its 160-bit output compared to MD5's 128-bit output makes it more secure from finding collisions and determining the original function. This is not to say that MD5 is insecure for Van Oorshcot and Wiener designed a brute force computer to find collisions in MD5. It took an average of twenty four days to find a collision [2]. Message Authentication Codes are message digest that use a key to encrypt the message providing a little more security.

Message digests in themselves offer no security. However, they are often used to verify that a message was not tampered with and are a key component in digital security.

2.4. Digital Signatures

Digital signatures are used to verify that information being received is from a specific person. The purpose of a digital signature is the same as a normal signature

in its purpose in authenticating whom the sender is. Digital signatures are based on calculating the message digest of a certain piece of data and encrypting it using the private key of an asymmetric algorithm. The encrypted message digest, public key, and the original message - encrypted or clear - are then sent to the receiving party. The receiving party then takes the public key and decrypts the hash. The hash for the message is then calculated and then compared to the hash that has been decrypted. If the hashes are equal then the message verification is successful; otherwise signature authentication has failed.

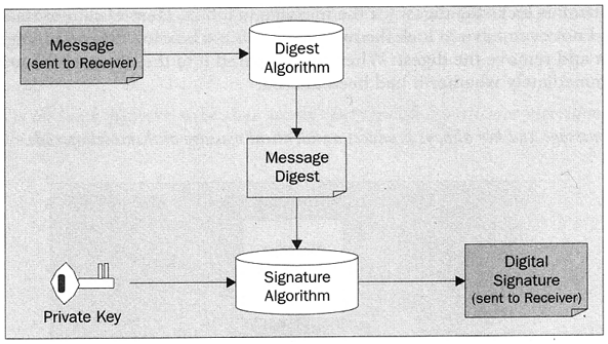

Figure 4: Digital signature process [4].

There are several reasons that the verification could have failed. It is possible that there was an error in the transmission so information wouldn't match. A greater threat to security is the message was intentionally altered by a middle man. In this case, the information in the message should be rejected since the information is not from the intended sender.

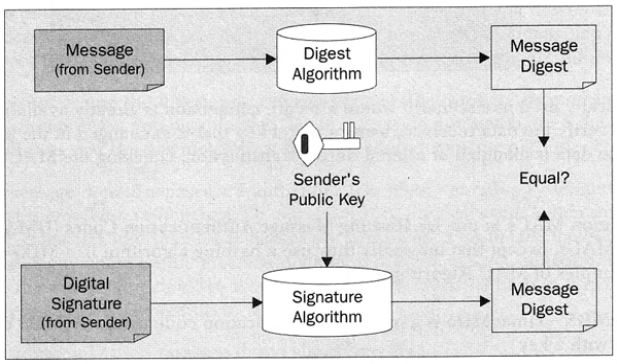

Figure 5: Digital signature verification [4].

Another issue that may arise is that someone could pretend to be an authorized sender by signing documents with their own key pair. This situation can be avoided by using certificates. Certificates contain information that verifies who a sender is. It is usually necessary to have a certificate authority, which is a trusted third party that issues certificates to those authorized to have them. Certificates created from a central authority make it much more difficult for someone to falsify someone else's identity.

2.5. JCE & JCA.

The Java Cryptography Architecture (JCA) and Java Cryptography Extension (JCE) are Sun's solution to being able to easily implement cryptographic services. Services provided by the two packages include Signatures, Ciphers, Message Digests, Message Authentication Codes, Key generators and more. The main convenience of the JCE and JCA are the ability to implement cryptographic services with minimal knowledge of how algorithms work. Most cryptographic algorithms are mathematically intensive and would be very time consuming to implement. The JCE and JCA take care of the underlying semantics making it easier to use cryptographic services. A key attribute to the cryptography architecture is the ability to use algorithms created by third-parties known as "providers." Providers follow specific guidelines set in the JCA and JCE that allow users to use algorithms with the same syntax regardless of the provider. Also, as new algorithms are developed, the user does not have to create or learn new code to implement the algorithm. For example, the standard Sun provider that comes built-in has very few services. The lack of services is easily solved by downloading a third-party, JCE compliant package that uses the same syntax as the Sun provider.

3. Implementation

Several actors were created in order to implement the various cryptographic services. A common feature between all the actors is that all information is passed as a byte array. The motivation for this is the cipher and signature classes act on byte arrays. For example, to allow for hybrid encryption it is necessary for the Key object to be in a byte array. Also, when reading in files, information must be read in as buffer of bytes and not String lines. The reason for this is it is possible end line characters to be output during encryption. This has no significance when reading from a byte buffer for decryption. However, if it were being read in as a String for decryption the end line character would be ignored because end line characters delineate separate Strings. Therefore data being decrypted is not the same as the original data, resulting in an unsuccessful decryption.

3.1. AsymmetricEncryption/Decrytption Actors

These actors implement asymmetric cipher algorithms following the JCE specifications that exist on the system. The user may specify the algorithm, provider, key size, mode and padding to be used. The AsymmetricDecryption actor is responsible for creating the public/private keys. The public key is then distributed to AsymmetricEncryption actors that encrypt the data to be sent. This data is sent to the AsymmetricDecryption actor that it received the public key from and is decrypted.

3.2. SymmetricEncryption/Decrytption Actors

These actors implement symmetric cipher algorithms following the JCE specifications that exist on the system. A user may specify the algorithm, provider, key size, mode and padding to be used. The SymmetricEncryption actor is responsible for creating the secret key. The SymmetricEncryption actor then encrypts the data using the secret key and sends the data and key to a SymmetricDecryption actor. The SymmetricDecryption actor receives the data and secret key and decrypts the message with the key. In certain instances, such as Cipher Block Chaining mode, information about the parameters used for encryption need to be sent. These parameters can be sent publicly without jeopardizing the security of a file.

3.3. SignatureSigner/Verifier Actors

These actors implement signature algorithms following the JCE specifications that exist on the system. These actors allow you to specify the algorithm and provider. The SignatureSigner actor takes the data and calculates the message digest for it. This digest is then encrypted using a private key created in the SignatureSigner actor. The encrypted message digest, original message and public key are then sent to a SignatureVerification actor. The SignatureVerification actor decrypts the encrypted message digest using the public key that it received. It calculates the message digest of the message received and compares the two digests. If the two digests are the same then the original message is sent out. If the digests are different, there was an error sending data or the original message was modified after it was signed and a message stating that the signature failed is displayed.

3.4. Bouncy Castle Cryptography Actors

These actors are an extension of the actors mentioned above that use the Bouncy Castle provider. The Bouncy Castle provider was chosen because it is freely distributed and contains a wide variety of algorithms. The purpose of this set of actors is the ability to use the cryptography services without needing to look at the provider specifications to determine information like what algorithms are padding schemes exist for the provider. Also, naming conventions for the same vary by provider. This set of actors removes this burden from the user promoting the use of cryptographic services.

3.5. Base Classes

Several base classes were created to take advantage of the inheritance properties of Java and to avoid writing redundant code. The main class for all actors is the CrypographyActor class. This class contains several helper classes to convert data types and partially addresses the retrieval of parameters and processing of data. Many of the declarations for the ports and parameters are made here. The SignatureActor and CipherActor implement more things that are shared by the respective base classes.

4. Conclusion

Implementing cryptographic actors allows for distributed models to be securely created in Ptolemy II. This gives Ptolemy II the features needed to safely implement distributed control systems and distributed computing systems. For example in the PATH project without secure modeling, computers could be tricked into crashing into each other if an unauthorized agent sent them the transmission. With security guards in place, such a transmission would be checked to see if it is an authorized sender to make sure it is not such an attack. The main plan in the future is to implement a certificate system for modeling purposes. This would make signature signing and verification much more secure. However, this can only be a model since the actual implementation would require the certificate authority to be a third-party entity.

5. Acknowledgments

First and most importantly I would like to thank my parents for they were the ones that brought me to this world and always supported me everything I've done. I would also like to thank Professor Edward Lee for giving me the opportunity to work on this project; all the graduate mentors who always willing to take time out of their busy schedules to help; the CHESS and SUPEB staff; Riya from Blake's on Telegraph; and Lindsay who did the final editing of this paper. And last but surely not least, the Terrible Terry Tate (aka TTT), office line backer.

6. References

[1] http://java.sun.com/j2se/1.4.1/docs/guide/security/CryptoSpec.html
[2] http://rsasecurity.com
[3] http://java.sun.com/j2se/1.4.2/docs/guide/security/jce/JCERefGuide.html
[4] http://www.cs.ttu.edu/~cs5331/ns/modules/