ICS

TECHNICAL REPORT

Security on Energy Level in the Bio-Networking Architecture

Technical Report 01-11
Department of Information and Computer Science
University of California, Irvine

Information and Computer Science

University of California, Irvine

Sanghoon Song and Tatsuya Suda

March 16, 2001

Security on Energy Level in the Bio-Networking Architecture

Technical Report 01-11

Department of Information and Computer Science
University of California, Irvine

Sanghoon Song and Tatsuya Suda

March 16, 2001

Abstract□ **Cyber-entities in the Bio-networking Architecture are autonomous mobile agents that construct network applications. The energy in the Bio-networking Architecture is a unit of exchange for service or use of resources and the behavior of a cyber-entity is influenced by its energy level. In this paper, we describe four types of attack on the energy level of a cyber-entity: malicious modification of the energy level by a cyber-entity itself, tampering the energy level of a migrating cyber-entity by a malicious node, malicious modification of the energy level by another cyber-entity, and malicious modification of the energy level by a platform. The energy level of a cyber-entity can be stored in itself, the platform in which the cyber-entity resides, or a centralized trusted node. The effective protection techniques depend on the location where the energy level is stored. We explore and discuss the security techniques and their limitations for three storage models: energy level in cyber-entity, energy level in trusted node, and energy level in platform.**

1. Introduction

Future network applications require scalability, adaptability, and survivability/availability. The Bio-Networking Architecture is a new architecture that enables future network applications [9]. The paradigm is inspired by the principles and mechanisms that allow biological system to scale, adapt, and survive. The Bio-

Networking Architecture consists of two major components, cyber-entities and Bio-net platforms.

Cyber-entities are autonomous mobile agents that construct network applications. Bio-net platforms are execution environments for the cyber-entities and supporting facilities for cyber entities. Each cyber-entity has basic functionality related to their application and follows a set of simple behavior rules, such as energy exchange and storage, death, migration, replication, and reproduction. Cyber-entities may gain energy in exchange for performing a service, and they may pay energy to use network and computing resources. The abundance or scarcity of stored energy may affect cyber-entity behaviors. Thus, the energy level of a cyber-entity may increase or decrease as a result of its behavior and protecting energy level is important.

We explore mainly security on energy level of a cyber-entity. There are four types of possible attacks on energy level: malicious modification of the energy level by a cyber-entity itself, tampering the energy level of a migrating cyber-entity by a malicious node, malicious modification of the energy level by another cyber-entity, and malicious modification of the energy level by a platform.

The energy level of a cyber-entity can be stored in itself, in the platform in which the cyber-entity resides, or in a trusted node. The protection techniques and limitations of each attack type depend on the location where the energy level is stored.

If the energy level of a cyber-entity is stored in itself, it is difficult to prevent the *malicious modification of the energy level by a cyber-entity itself*. If a trusted node keeps the energy level, it examines the transaction information from the counter part of the transaction to detect any malicious attempt by a cyber-entity itself. If the energy level is stored in a platform, a cyber-entity cannot modify its energy level intentionally.

The *tampering the energy level of a migrating cyber-entity by a malicious node* can be avoided using ordinary secure connection, such as Secure Socket Layer (SSL), if the energy level is stored either in cyber-entity or in platform. If a trusted node keeps the energy level, the energy level is safe from tampering attack since it is not transferred during migration.

The *malicious modification of the energy level by another cyber-entity* can be avoided by the inherent JAVA security and by the mechanism of encryption in object serialization. However, if the energy level is stored in a trusted node, an authentication mechanism is required to prevent any masquerade attack by a malicious cyber-entity.

As for *malicious modification of the energy level by a platform*, any of three cases cannot provide an effective protection technique in conventional computer systems. A secure infrastructure is required to prevent a malicious modification by a platform, allowing cyber-entities to run securely.

In Section 2, we describe an overview of the Bio-Networking Architecture. The security techniques on mobile agents are described in Section 3. Section 4 describes the four types of attack on the energy level of a cyber-entity, and Section 5 describes protecting techniques for three storage models. Finally, we give the discussion and conclusion in Section 6.

2. Bio-Networking Architecture

In the Bio-Networking Architecture, applications are constructed using multiple autonomous biological entities, called cyber-entities. The desirable characteristics of applications, such as scalability, adaptability, and survivability/availability, emerge from multiple interacting cyber-entities.

Cyber-entities collect energy from human users or other cyber-entities, but they must also give energy to Bio-net platforms in order to run. Cyber-entities also try to maximize their energy gain from users while minimizing energy expenditures to the Bio-net platforms. If a cyber-entity exhausts its energy, it will not be allowed to run by the Bio-net platform, i.e., it dies of starvation. Because energy plays an important role in the lives of cyber-entities, cyber-entities are forced to adapt to user demand and the network environment by energy considerations.

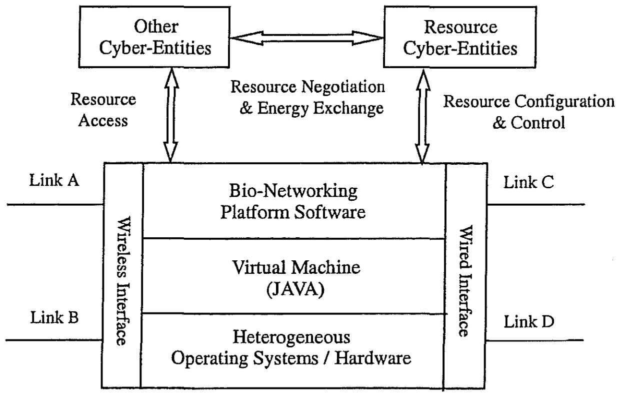

Other Cyber-Entities
Resource Cyber-Entities
Resource Access
Resource Negotiation & Energy Exchange
Resource Configuration & Control
Link A
Link B
Link C
Link D
Wireless Interface
Wired Interface
Bio-Networking Platform Software
Virtual Machine (JAVA)
Heterogeneous Operating Systems / Hardware

Figure 1. Bio-Networking Node

Natural selection occurs because cyber-entities with no energy will die of starvation, and cyber-entities with large amount of energy have more opportunities to reproduce. Therefore, if a cyber-entity's behavior enable it to collect more energy than other cyber-entities, it will live longer and give birth to more cyber-entities similar to itself. Over time, the cyber-entities that comprise an application will evolve to behaviors that are more useful and/or efficient.

A Bio-net platform provides an environment for cyber-entities to run. A Bio-net platform is any networked hardware device that runs a Java Virtual Machine (JVM) and the Bio-Networking platform software. All resources on a Bio-net platform, such as CPU time, memory, disk space, network bandwidth, must be purchased with energy units. When a cyber-entity is created, it is given energy units by the system administrator who created it or by its parent cyber-entities. Cyber-entities use their energy to buy resources from Bio-net plat-forms. If a cyber-entity exhaust its energy, it will not be allowed to run on a platform and all resources that are allocated to the cyber-entity will be freed.

The platform software also provides system level services that cyber-entities cannot perform directly, such as migration and reproduction. Since these services consume CPU and network resources, cyber-entities must pay energy to the platform to receive these services.

3. Security techniques on mobile agent

In this section, we will describe existing security techniques for mobile agent systems and their inherent limitations. Security techniques for mobile agent include protecting hosts from malicious mobile agents and protecting mobile agents from malicious hosts.

3.1 Protecting hosts from malicious agents

In this section, we describe briefly functions and their limitation of the techniques that can be used to protect hosts from malicious mobile agents. Techniques described include *access control* and *code verification.*

Access Control - An obvious approach to protect a host from a malicious mobile agent is to control the access request of mobile agents. A Mobile agent can be executed in a protected boundary where it cannot damage anything that is not allowed to do. Some mobile agent systems use a reference monitor to control the access request based on the level of authorization for a mobile agent. The reference monitor restricts access to limited resources based on a security policy. The host determines a security policy based on the identity of mobile agent's author and of its sender. The identity of mobile agent's author can be verified using authentication techniques in distributed systems. However, some of the known difficulties in conventional authentication systems are more serious in mobile agent systems [1].

CodeVerification – An alternative technique is for the host to determine that the mobile agent is safe to execute [3]. The key idea is to attach to a mobile agent an easily checkable-proof that its execution does not violate the safety policy of the receiving host. The proof is encoded in a form that can be transmitted digitally to the receiving host and then quickly validated using a simple proof-checking process. A central component of proof-carrying code implementation is the *safety policy* that the receiving host desires to enforce for any untrusted mobile agent. The safety policy is used not only by the receiving host, but is also exported to the originator of a mobile agent. Proof-carrying code requires a receiving host to understand the vulnerabilities of its system thoroughly.

3.2 Protecting mobile agents

Protecting mobile agents from a malicious host is essential for agent-based systems. Attacks by a malicious host such as denial of service, random modifications of mobile agent code or of its output, and replay, are difficult to prevent. In the following, we describe briefly the existing techniques such as *Computing with Encrypted Function* and *Limited Blackbox Security*, including *Secure Coprocessor.*

Computing with encrypted functions – Sander and Tschudin proposed a mobile agent protection technique that requires a computer to execute a cipherprogram without understanding it [5]. The goal is to encrypt functions such that the encrypted functions can be implemented as programs. The resulting program will consist of cleartext, not cipher text, instructions that a processor understands. However, the processor will not be able to understand the function of the program. For example, let us consider a linear function A and compute the encrypted function $B := S \cdot A = E(A)$ by selecting at random an invertible matrix S. Let the program P implement the encrypted function B with cleartext instructions. Then, the program P will output the garbled result B(x). Since the encrypted function B(x) garbles the output of the computation, we can obtain the result A(x) by computing $S^{-1} \cdot B(x)$ for input x. The challenge is to find encryption schemes S for arbitrary functions.

Limited blackbox security – An agent is considered to be a blackbox if only its input and output behavior is observed, and if the agent code may not be modified by read or modification attacks. There is no known method that provide complete blackbox security. Hohl, however, presented a technique that provides blackbox security only for a certain known time interval [2]. In order to fulfill the limited blackbox property, Hohl proposed several conversion algorithms. A conversion algorithm generates a new agent from a given agent, which differs in code and representation but yields the same results and is hard to analyze. The generated code must be constructed in a way that program analyzing techniques cannot be used to analyze the agent before the expiration date has passed.

The conversion algorithms are called *obfuscating* or *mess-up* algorithms. One of the mess-up algorithms called *variable recomposition* takes the set of program variables, divides each variable content into segments and creates new variables that contain a recomposition of the original segments. Each code segment to access the original variable is adapted correspondingly. They presented two other mess-up algorithms called *conversion of control flow elements into value-dependent jumps* and *deposited keys.*

The protection strength of the blackbox security mechanism depends on the structure and attributes of the used mess-up algorithms. The main problem is how long a protection interval has to be in order to fulfill a useful task. The even bigger problem is the question on how to determine these protection intervals from the used conversion algorithms.

Secure coprocessor – As an engineering solution, Yee proposed a secure coprocessor to provide a trusted execution environments for mobile agents [10]. A Secure coprocessor is a hardware module containing a CPU, bootstrap ROM, and secure non-volatile memory. This hardware is shielded from physical attacks. The privacy and integrity of the secure non-volatile memory provide the foundations for a secure system. If we are able to run a single trusted program on the system, we can use the program to verify the integrity of the rest of the system. To verify the integrity, a secure coprocessor maintains a tamper-proof database (kept in secure non-volatile memory) containing a list of the host's system programs along with their cryptographic checksums (hash functions). This technique, however, requires building a tamper-proof database for each host and building a secure coprocessor for each host

4. Types of attack on energy level in the Bio-net

Cyber-entities have several different behaviors such as migration, reproduction, service, and death behavior. Since most of the cyber-entity's behaviors are related to its own energy level, it is critical to protect energy communication mechanisms. We explore four possible types of attack on the energy level of a cyber-entity in Bio-net: malicious modification of the energy level by a cyber-entity itself, tampering the energy level of a migrating cyber-entity by a malicious platform, malicious modification of the energy level by another cyber-entity, and malicious modification of the energy level by a platform.

4.1 Malicious modification by a cyber-entity itself

Since energy is vital to cyber-entity life, cyber-entity must gain energy in order to survive by providing services to users or other cyber-entities. When a cyber-entity cannot provide services to users or other cyber-entities due to lack of demand, it may die of starvation. A malicious cyber-entity may intentionally modify its energy level in order not to die of starvation when demand is low.

If a cyber-entity can directly access its energy value, it is very difficult to prevent this type of attack. We may need to store the energy level of cyber-entities in the place, either in a platform or a trusted node, that cannot be accessed directly by a cyber-entity.

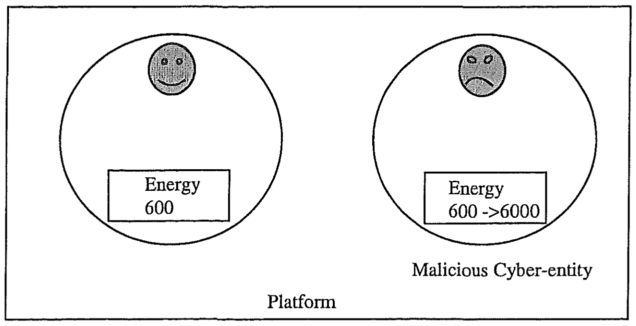


Figure 2. Malicious modification by a cyber-entity itself

4.2 Tampering by a malicious platform

Cyber-entities may migrate toward the source of demand when they have enough energy to pay migration and resource cost in destination. Since the TCP/IP protocol suite is vulnerable to a variety of attacks during packet transfer, a malicious platform of a competing service provider may intercept a migrating cyber-entity and modify its energy value, eliminating competing cyber-entities.

We may use a cryptography technique to protect from tampering of energy level during migration. In a more conservative way, we may need to store the energy level in a trusted node and avoid transferring of energy level.

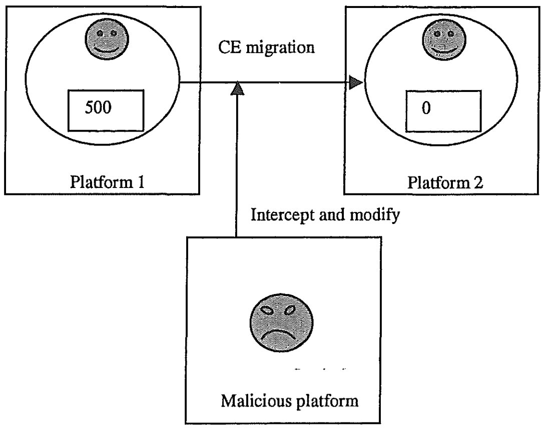


Figure 3. Tampering by a malicious platform

4.3 Malicious modification by another cyber-entity

Since multiple cyber-entities that comprise different applications can share a platform, a malicious cyber-entity in the same platform may intentionally modify the energy level of another cyber-entity that comprises an application of a competing service provider. This attack can be prevented if we assume the platform and cyber entities are implemented in Java, since the memory space for each cyber-entity and platform is protected from each other by the inherent Java security. In Java, programs cannot access arbitrary memory location, since Java does not have the notion of pointer type. In a language like C++, however, a program can manipulate a pointer to access a private data of another class [4], and thus, some protection mechanism is necessary.

However, there exists a threat of modification of energy level by another cyber-entity when a cyber-entity object is serialized for migration. Object serialization is a feature of Java that allows an object to be written to a file as a series of bytes in order to transfer to another host. In practice, it is much easy for other cyber-entities to change the data in a binary file. To prevent this type of attack, we may use existing techniques to make object serialization secure. For instance, if a variable is declared as *private transient*, the variable will not be written out in object serialization. After all non-transient data being written out, the transient data can be written out in any encrypted format. Thus, the *private transient* keyword associated with a variable would not allow other cyber-entities to read or write that particular variable in object serialization [4].

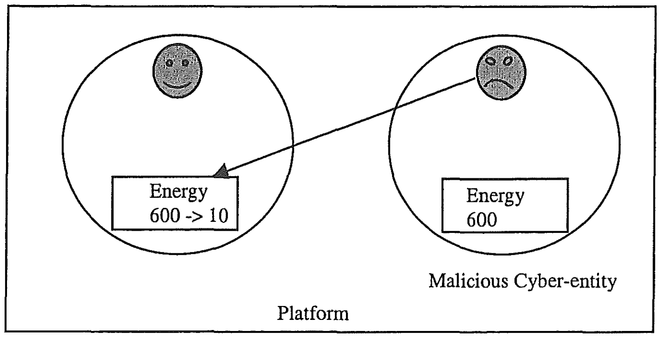


Figure 4. Malicious modification by another cyber-entity

4.4 Malicious modification by a platform

A malicious platform may intentionally modify the energy level of a cyber-entity. Since a platform has full access to cyber-entity, it is practically impossible to protect from a malicious platform in conventional computer systems [1]. The platform can manipulate cyber-entity behavior and state, replay the behavior, or return wrong results to the system calls issued by a cyber-entity. We may circumvent the problem by installing a platform only in trustworthy parties or by using specialized, tamper-resistant hardware such as *secure coprocessor* to ensure the integrity of a platform.

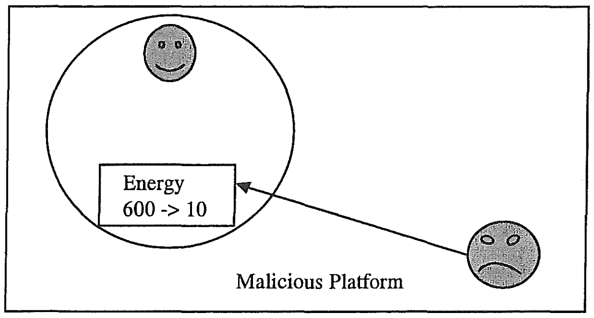


Figure 5. Malicious modification by a platform

5. Protecting energy level

We consider three places which can store the energy level: *cyber-entity itself, a trusted node,* and *platform* *in which a cyber-entity resides*. The effective protection techniques for energy level depend on the place where the energy level is stored. The following subsections describe the security techniques and their limitations for these three models. Section 5.4 describes the secure platform using coprocessor.

5.1 Energy level in cyber-entity

In this model, the energy level of a cyber-entity is stored in itself, and the cyber-entity modifies its energy level. When a cyber-entity migrates to another platform, it carries its energy level with itself. For each energy-related transaction, a cyber-entity modifies its energy level by a legitimate amount.

Since cyber-entities have total control over their energy level, a cyber-entity may maliciously increase its energy level without actually providing services to users or other cyber-entities in order not to die of starvation. In this case, it is difficult to protect the energy level from malicious modification by a cyber-entity itself.

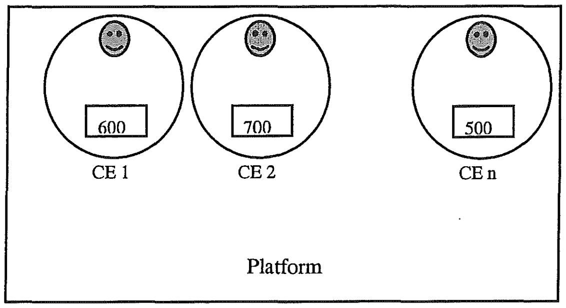


Figure 6. Energy level in cyber-entity

The second type of security attacks described in Section 4.2 (i.e., tampering by a malicious platform) can be prevented by an ordinary secure connection, such as SSL or any public key cryptography which can encrypt the data and verify their integrity.

The third type of security attacks described in Section 4.3 (i.e., malicious modification by another cyber-entity) can be prevented by using an existing technique of secure object serialization. The *private transient* keyword associated with a variable would not allow other cyber-entities to read or write that particular variable in object serialization. Also, The inherent JAVA security protects the energy level of a cyber-entity from malicious modification by another cyber-entity on the same platform, since the memory space of each

JAVA object is protected from each other.

As described in Section 4.4 (i.e., malicious modification by a platform), platforms can have total control over cyber-entities, manipulate the cyber-entity code and state, and replay some behaviors. It is practically impossible to protect the energy level from malicious modification by a platform. This necessitates the need for a secure infrastructure such as secure coprocessor to prevent a malicious platform attack.

As described above, keeping energy level in cyber-entity is vulnerable to malicious modification by a cyber-entity itself. To prevent tampering by a malicious platform, we need a secure connection mechanism, such as SSL or a public key cryptography that can maintain the integrity of an energy level. To prevent malicious modification by another cyber-entities in the same platform, we need to encrypt the energy level data in object serialization. A secure infrastructure is required to prevent a malicious modification by a platform.

5.2 Energy level in trusted node

In this model, a trusted node keeps energy levels of all cyber-entities that are active in the Bio-Networking Architecture. The trusted node is responsible for updating energy levels of all cyber-entities. When a cyber-entity migrates to another platform, it does not carry its energy level with itself. Since the trusted node has no information regarding energy-related transactions of cyber-entities, cyber-entities are required to send energy-related transaction information to the trusted node so that it can update the energy levels of cyber-entities.

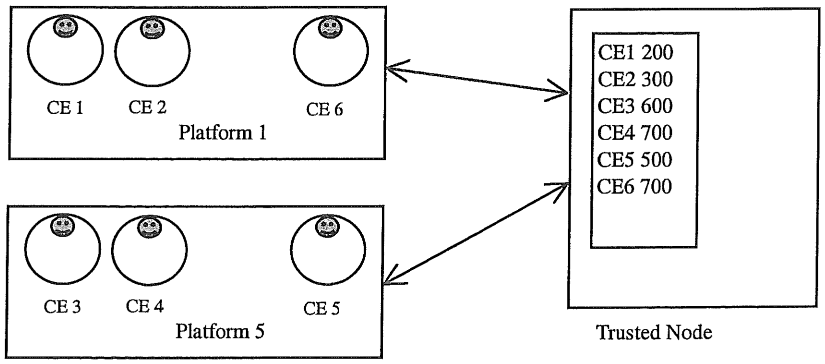


Figure 7. Energy level in trusted node

Since the energy level is stored in a trusted node and modified only by the trusted node, a cyber-entity does not have access to its own energy level and cannot maliciously modify its own energy level. However, the cyber-entity may intentionally provide incorrect energy-related transaction information to the trusted node, leading to an inaccurate energy level.

To protect from this type of attacks, the trusted node may request transaction information from the counter part of the transaction or the counter part may automatically send this information to the trusted node. If there is any discrepancy in transaction information, an attempt to maliciously modify the energy level is detected.

As for the type of security attacks described in Section 4.2 (i.e., tampering by a malicious platform), there does not exist a threat of tampering the energy level of a migrating cyber-entity by a malicious platform. However, in the scheme described above, each cyber-entity sends energy-related transaction information to the trusted node to update its energy level. Thus, there exists a threat of tampering attacks during transfer of energy-related transaction information. As in previous model, an ordinary secure connection, such as SSL, may be used to effectively avoid tampering attacks during transfer of energy-related transaction information.

Since energy level is kept in a trusted node, the type of security attacks described in Section 4.3 (i.e., malicious modification by another cyber-entity) is not possible. A malicious cyber-entity cannot directly attacks energy level of other cyber-entities on the same platform. However, a malicious cyber-entity may masquerade as a trusted one. To prevent a malicious cyber-entity from masquerading as a trusted cyber-entity, cyber-entities need to be authenticated by the trusted node. However, the authentication of a cyber-entity is not easy because the state of a cyber-entity will vary over its lifetime.

As for the type of security attacks described in Section 4.4 (i.e., malicious modification by a platform), platforms cannot directly attack energy level of cyber-entities. However, if there is a malicious platform involved in an energy-related transaction, examining the energy-related transaction information from cyber-entities involved in the transaction does not lead to the detection of malicious attempts. This is because of the following. Since a platform can modify cyber-entity's code to collaborate with the platform, it can collect more energy than what the cyber-entity has to give. This necessitates the need for a secure infrastructure such as secure coprocessor to prevent a malicious platform attack.

As described above, keeping energy level in trusted node can avoid malicious modification by a cyber-entity itself by checking the counter part of a transaction. Secure connection mechanism may be used to effectively avoid tampering attacks during transfer of energy-related transaction information. The authentication of a cyber-entity is required to prevent a malicious cyber-entity from masquerading as a trusted cyber-entity. A secure infrastructure is required to prevent a malicious platform attack.

5.3 Energy level in platform

In this model, a platform keeps energy levels of all cyber-entities that reside in the platform. Platforms are also responsible for updating the energy levels of cyber-entities. When a cyber-entity migrates to another platform, the cyber-entity is serialized first, and then, the platform appends the energy level to the serialized cyber-entity body. Then, the cyber-entity is transferred to the destination platform. Each platform is assumed to know all energy-related transactions of each cyber-entity that resides in the platform, in order to update energy levels.

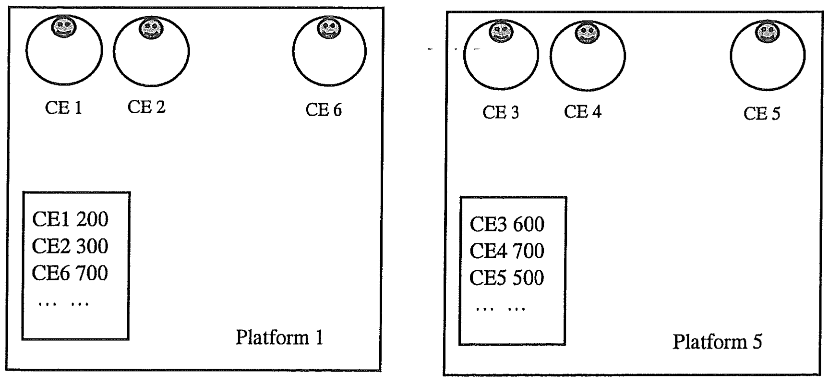


Figure 8. Energy level in platform

The memory space for each cyber-entity and platform is protected from each other by the inherent Java security. Since all energy levels that are stored in a platform are protected from access by cyber-entities, the first type of attacks described in Section 4.1 (i.e., malicious modification by a cyber-entity itself) is not possible. Also, the third type of attacks described in Section 4.3 (i.e., malicious modification by another cyber-entity, respectfully) is not possible.

The second type of security attacks described in Section 4.2 (i.e., tampering by a malicious platform) can be prevented by an ordinary secure connection, such as SSL which can encrypt the data and verify their integrity.

As for the type of security attacks described in Section 4.4 (i.e., malicious modification by a platform), a platform can maliciously modify the energy level of a cyber-entity. Thus, a secure infrastructure for cyber-entities using secure coprocessor is needed to prevent a malicious platform attack.

As described above, keeping energy level in platform is vulnerable only to malicious platform attack. The malicious modification by a cyber-entity itself and the malicious modification by another cyber-entity are prevented by inherent JAVA security. Secure connection mechanism may be used to effectively avoid tampering the energy level of a migrating cyber-entity by a malicious platform.

5.4 Secure platform using a secure coprocessor

We can build a secure platform using secure processor architecture proposed by Yee[10]. Secure coprocessor is a hardware module that has a CPU, bootstrap ROM, and secure non-volatile (battery backed) memory. This secure processor can be implemented as a circuit board on the system bus. The hardware module is designed to protect from physical attack. The only way to access the internal state is through the I/O interface to the module. It can store cryptographic checksums and keys safely. The hardware can include a high speed DES engine to accelerate encryption and decryption function.

The hardware must be packaged so that any physical attempt to break the hardware module will erase the data kept in the secure non-volatile memory. An attacker may see the hardware structure of the secure coprocessor, but the attacker cannot read nor modify the secure non-volatile memory of the secure coprocessor except through normal I/O interface with proper user authentication.

To detect any physical attack, the hardware module has special sensing circuitry. The sensing circuitry erases the secure non-volatile memory before an attacker disables the sensors or reads memory contents. For example, to prevent direct attack, we can use sensors consisting of fine nichrome wire and low power sensing circuits powered by a long-lived battery. The wires are densely wrapped in many layers around the circuit board and the entire board is then dipped in epoxy. The sensing circuits can detect open circuits or short circuits in the wires. Physical attack by mechanical means cannot penetrate the epoxy without breaking one of these wires. To prevent the attack of dissolving the epoxy with solvents to expose sensor wires, the epoxy is designed to be chemically stronger than the sensing wires. More techniques to prevent other types of physical attacks are described in [10].

With a strong security base available in the form of secure coprocessor, we can provide the trust in the integrity of a platform's system software (operating system kernel, system related user-level software such as

Bio-networking Architecture platform software). If we are able to run a single trusted program on the system, we can verify the integrity of the rest of the system by using that program. The secure processor enables running an initial trusted program and this software verifies the integrity of the platform's system program resident on the disks.

For verification process, a secure coprocessor keeps a tamper-proof database in secure non-volatile memory. The database contains a list of the platform's system programs and their cryptographic checksums. The cryptographic checksums cannot be forgeable: given a file f_1 and the cryptographic checksum function *crypt_checksum* (), creating a program f_2 such that

$$f_1 = f_2 \text{ and } crypt\_checksum(\mathrm{f}_1) = crypt\_checksum(\mathrm{f}_2)$$

is computationally intractable. Among the many cryptographic checksum functions, the Karp-Rabin fingerprint functions are particularly attractive, since they are fast and easy to implement.

The secure coprocessor make sure the system boots securely. After secure booting, the secure coprocessor can help the platform operating system by providing security services, such as verification of integrity for any stored data and encryption/decryption function.

6. Discussion and conclusion

Table 1 shows the summary of the security on energy level in three storage models: cyber-entity, trusted node, and platform. The attack type of malicious modification by a cyber-entity itself is difficult to avoid in the model of keeping energy level in cyber-entity. In the model of keeping energy level in a trusted node, the trusted node examines transaction information from counterpart of a transaction to detect any intentional modification by a cyber-entity. This model will increase the network traffic due to the transfer of each transaction information. However, keeping energy level in platform effectively prevents malicious modification by a cyber-entity.

The tampering during migration can be avoided using ordinary secure connection such as SSL in the models of keeping energy level in cyber-entity or keeping energy level in platform. Keeping energy level in trusted node is safe from tampering attacks, since energy level information is not transferred during migration.

In the model of keeping energy level in cyber-entity, the attack type of malicious modification by another cyber-entity can be avoided by the inherent JAVA security and by the mechanism of encrypting a sensitive data in object serialization. In the model of keeping energy level in trusted node, an authentication mechanism is required to prevent masquerade attacks by a malicious cyber-entity. However, JAVA security is enough to prevent the malicious modification by another cyber-entity in the model of keeping energy level in platform.

Table 1. Security on energy level in three storage models

	In cyber-entity	In trusted node	In platform
Malicious modification by a cyber-entity itself	Difficult to prevent	Checking counterpart Secure connection	JAVA security
Tampering during migration	Secure connection	Safe	Secure connection
Malicious modification by another cyber-entity	JAVA security Encrypt in object serialization	Requires authentication	JAVA security
Malicious platform	Secure coprocessor	Secure coprocessor	Secure coprocessor

All three models have difficulty in protecting energy level from malicious modification by a platform in conventional computer systems. A secure infrastructure for cyber-entities using secure coprocessor is needed to prevent a malicious platform attack. Considering the malicious modification by a cyber-entity may be the most probable attack to energy level in Bio-net application, it is not safe to keep the energy level in cyber-entity.

In the model of keeping energy level in a trusted node, the overhead of authenticating cyber-entities and information transfer of each energy-related transaction may cause the trusted node to become a processing bottleneck and create a heavy congestion around the trusted node, limiting the scalability of this solution. A set of trusted nodes might be distributed across the network to alleviate the bottleneck.

However, keeping energy level in platform effectively prevents the attack of malicious modification by a cyber-entity. Considering a secure infrastructure for cyber-entities using secure coprocessor is required, it is most safe and efficient to keep energy level in platform. The only conceivable overhead compared to the other models is maintaining the energy level table in a platform. The table should be constructed in a way to insert, delete, and search an item efficiently. If the size becomes fairly large, a hash table will show a good performance.

References

[1] D. M. Chess, "Security Issues in Mobile Code", in G. Vigna (Ed.), Mobile Agents and Security, Lecture Notes in Computer Science1419, Springer-Verlag, Berlin, 1998, pp1-14

[2] F. Hohl, "Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts", in G. Vigna (Ed.), Mobile Agents and Security, Lecture Notes in Computer Science1419, Springer-Verlag, Berlin, 1998, pp92-113

[3] G. C. Necula, P. Lee, Safe, "Untrusted Agents Using Proof-Carrying Code", in G. Vigna (Ed.), Mobile Agents and Security, Lecture Notes in Computer Science1419, Springer-Verlag, Berlin, 1998, pp61-91

[4] S. Oaks, "JAVA Security", O'Reilly, 1998

[5] T. Sander, C. Tschudin, "Protecting Mobile Agents against Malicious Hosts", in G. Vigna (Ed.), Mobile Agents and Security, Lecture Notes in Computer Science1419, Springer-Verlag, Berlin, 1998, pp44-60

[6] D. Stinson, "Cryptography: Theory and Practice", CRC, 1995

[7] W. Stallings, "Cryptography and Network Security", Prentice Hall, 1998

[8] Tatsuya Suda, http://netresearch.ics.uci.edu/bionet/

[9] Michael Wang, Tatsuya Suda, "The Bio-Networking Architecture: A Biologically Inspired Approach to the Design of Scalable, Adaptive, and Survivable/ Available Network Applications" TR 00-03, February 2000, Department of Information and Computer Science, UC-Irvine

[10] Bennet Yee, "Using Secure Coprocessor" CMU-CS-94-149, May 1994, School of Computer Science, Carnegie Mellon University

[11] A. Young, M. Yung, "Sliding Encryption: A Cryptographic Tool for Mobile Agents", Proc. 4th Int'l

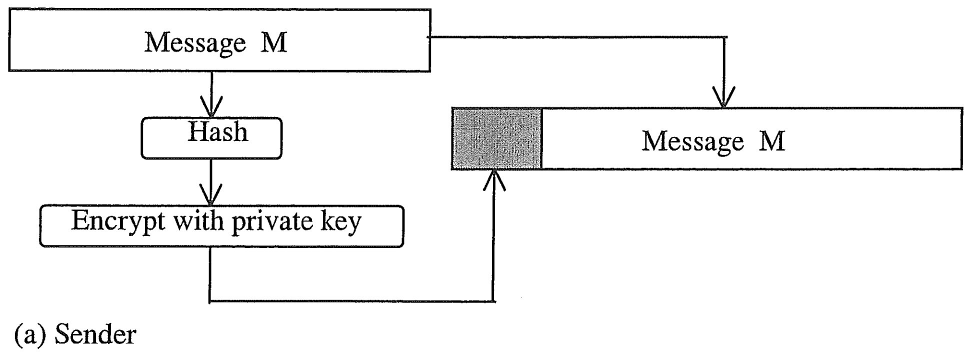
Wksp, Fast Software Encryption, FSE

Appendix

1. Digital Signature

Digital signature provides authentication and integrity capability. A signature ensures that a message is originated from a certain person and it has not been tampered with. Signatures do not provide confidentiality. A message is sent as a plaintext accompanied with a signature. We need to encrypt the message to provide confidentiality.

Secret-key cryptography (shared key encryption or conventional encryption) uses one key that is shared by the sender and the receiver. The sender encrypts a message using the shared key, and the receiver decrypts the encrypted message using the same key. The secret-key cryptography is 100 to 1000 times faster than the public-key cryptography. However, the secret-key encryption requires a secure key management system to distribute secret keys to users effectively.



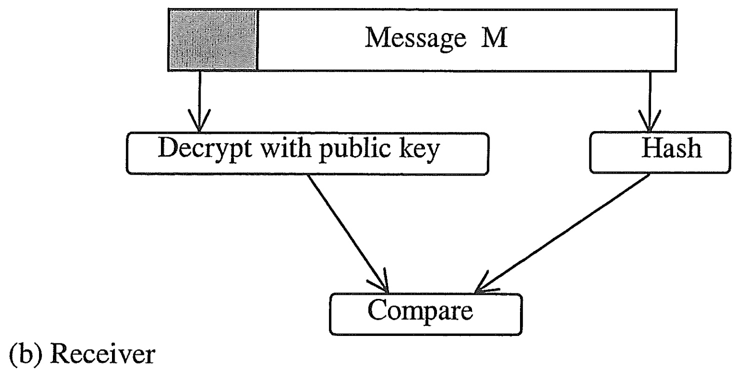


Figure 9. Digital signature using public-key cryptography

Public-key cryptography uses a pair of keys, private key and public key, to encrypt and decrypt messages.

The private key should be kept in secret, whereas the public-key can be distributed to the public. Each of these keys can be used for both encryption and decryption, however, the message encrypted with one key can be decrypted only with the other key. The disadvantage of public key encryption is its slow speed.

Public key cryptography can be used to sign a message. A sender encrypts a message with the sender's private key. Any receiver can decrypt the message using the sender's public key. The receiver know the message must have come from the sender since the sender is the only one who has the corresponding private key to the public key that the receiver used to decrypt the message. Since the public key cryptography is slow, it is more efficient to sign the hash value (cryptographic checksum) for a message than to sign the entire message.

To distribute public keys reliably and efficiently, a trusted third party called certification authority (CA) issues a certificate that contains the name of the party and its public key. Then the CA signs the certificate with its private key and gives it to the participant with the matching private key. A participant can send its certificate to other participants. Other participants can verify that the certificate was created by the CA using the CA's public key.

A hash function produces a fixed size hash value H(M) for a given variable size message M. The hash value is encrypted with the sender's private key and is appended to the message M. The receiver decrypts the encrypted hash value with the sender's public key and compare it with the hash value of the received message to check integrity. The following illustrates a way of digital signature using public-key cryptography and hash value.

2. SSL(Secure Socket Layer)

Secure Socket Layer (SSL) is designed to make use of TCP to provide a reliable end-to-end secure service. It provides authentication, message integrity, and confidentiality. Since SSL is at the socket layer and is independent of the higher level application, it can provide security services to higher level protocols such as TELNET and FTP. SSL consists of two layers of protocols as shown in figure 10.

The SSL Record Protocol provides basic security services to upper layer protocols. The SSL Handshake Protocol among three upper layer protocols is used to negotiate security parameters for an SSL connection.

The other two protocols in the upper layer protocols, Change Cipher Spec Protocol and Alert Protocol, are supplementary for SSL.

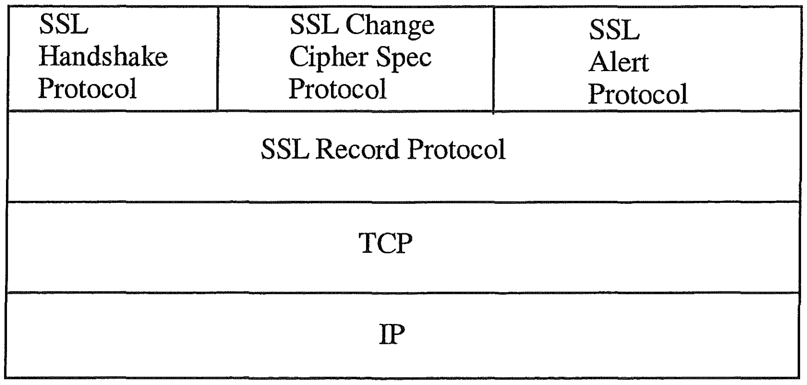


Figure 10. SSL protocol Stack

The SSL Record Protocol fragments the application message into blocks of 214 bytes or less and optionally compresses the data. Then, the MAC (message authentication code) over the compressed data is added. Next, the compressed message with the MAC is encrypted using symmetric encryption scheme. The final step is to add a header consisting of content type, major version, minor version, and the length of the record.

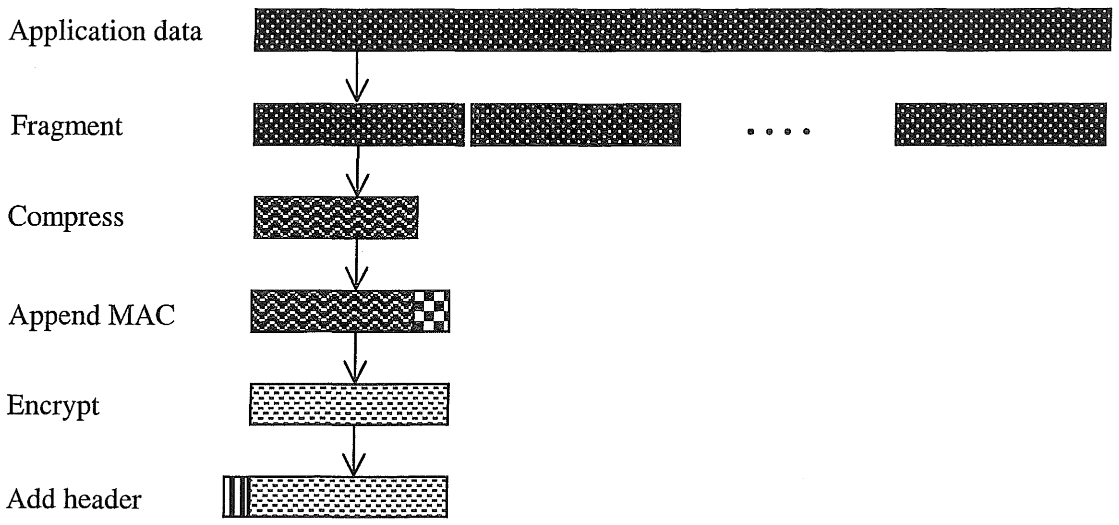
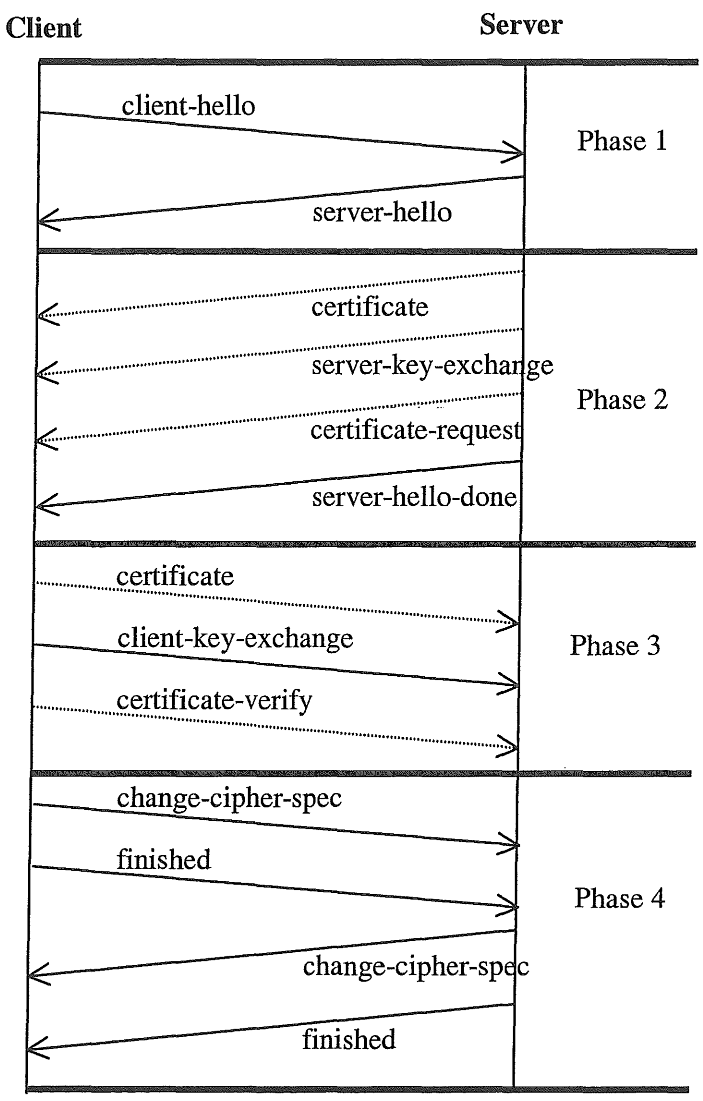


Figure 11. SSL Record Protocol steps

The Handshake Protocol allows the server and client to authenticate each other and to negotiate an encryption algorithm, MAC algorithm, and cryptographic keys. This protocol is used before any application data is transmitted. The Handshake Protocol consists of four message exchange phases, described next.



* Dotted arrows are optional or situation-dependent.

Figure 12. Handshake Protocol timing

The first phase establishes security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random number. This phase is to initiate a logical connection and establish the security capabilities associated with it.

The second phase is for server authentication and key exchange. The server optionally sends its certificate, a key-exchange message, and a certificate-request message. The final message, always required, is the server-hello-done message. This is to indicate the end of server hello and associated messages.

The third phase is for client authentication and key exchange. The client sends a certificate message if the server requested a certificate at the second phase. Then, the client-key-exchange message must be sent. Finally the client optionally sends a certificate-verify message to provide explicit verification of a client certificate.

The fourth phase completes the setting up a secure connection. The client sends a change-cipher-spec message and a finished message. In response these two messages, the server send its change-cipher-spec message and a finished message.

After the fourth phase, the handshake is complete and the client and server may start to exchange application data.