Networking Aspects for the Security of Game Input

Christos Bouras, Vassilis Poulopoulos and Vassilis Tsogkas
Research Academic Computer Technology Institute,
N. Kazantzaki, Panepistimioupoli Patras, 26500, Rio, Greece
bouras@cti.gr, poulop@cti.gr, tsogkas@cti.gr
tel. +302610996951
fax. +302610996358

***Abstract*— Following the trends of our era, Games At Large IST Project introduces an innovative platform for running interactive, rich content multimedia applications over a Wireless Local Area Network. Games@Large project's vision is to provide a new system architecture for Interactive Multimedia which will enhance existing CE devices such as, Set Top Boxes (STB), Small Screen and other devices, which are lacking both the CPU power and the graphical performance to provide a rich user experience. This paper presents the input command transferring module that provides encryption capabilities for ensuring the security of the transmitted sensitive user data. In a nutshell, the client software when capturing input from a keyboard device encrypts the commands that are transmitted over a Wireless Local Area Network, using a public key encryption scheme and the server is responsible for decrypting and executing the commands to the corresponding game application.**

***Index Terms* — online gaming, remote command execution, input device capturing, asymmetric encryption, reverse channel**

I. Introduction

Computer games constitute nowadays one of the most dynamic and fastest changing technological area, both in terms of market evolution and technology development. In this area, as computer games are evolving and online activities and gaming become part of our lives, the need of interaction within a client – server architecture becomes very intense. The successful paradigms of online gaming such as WoW [6], Half Life [7] and Second Life [8] are only just the beginning of a new era for online games. The idea that lies behind online gaming is that a game that can be played by multiple users should not have only a local context. The basic game software is installed on the client machine while multiple servers are assigned with the task of interconnecting all the possible users to what is called the "world" or the scenario of the game. Games at Large project goes one step further than the classical procedure of online gaming and the main intention is to enhance the idea of application on demand [5] in order not only to support games on demand, but also to enable devices that lack the physical power to load a game, to run games [9].

The main idea is that one or more powerful servers will actually execute the game for the client and only frame screens of the game, and not the game loader or the execution of complex graphics, will be presented to the client. On the other hand the basic aspect of a game is the interaction with the end user (gamer). This means that apart from only presenting the game to the user (through this client – server architecture) the system must be able to capture the input from any input devices of the end user and transfer it to the server in order to represent the interaction that is done on a physical level when playing a game. An important aspect of the aforementioned procedure is security in the transferring of input commands. In particular, keyboard input, which in most cases depicts user sensitive data such as passwords or credit cars numbers, must be foolproof.

The object of secure communications has been to provide privacy or secrecy, i.e., to hide the contents of a publicly exposed message from unauthorized recipients. The asymmetric encryption / decryption channel solves the major confidentiality issue of secure communications. Cryptosystems [1] are symmetric if either the same piece of information (key) is held in secret by both communicants, or else that each communicant holds one from a pair of related keys where either key is easily derivable from the other. These secret keys are used in the encryption process to introduce uncertainty (to the unauthorized receiver), which can be removed in the process of decryption by an authorized receiver using his copy of the key or the "inverse key." This means, of course, that if a key is compromised, further secure communications are impossible with that key. On the other hand, in asymmetric cryptographic schemes the transmitter and receiver hold different keys at least one of which it is computationally infeasible to derive from the other.

The work on public key cryptographic systems has been rather intense over the last 20 years. The main difficulty in developing secure systems based on public key cryptography is not the problem of choosing appropriately secure algorithms or implementing those algorithms [3]. Rather, it is the deployment and management of infrastructures to support the authenticity of cryptographic keys: there is a need to provide an assurance to the user about the relationship between a public key and the identity (or authority) of the holder of the corresponding private key. In a traditional Public Key Infrastructure (PKI), this assurance is delivered in the form of certificate, essentially a signature by a Certification Authority (CA) on a public key [2].

In this paper we present the general architecture of the encryption subsystem which ensures that the input from any keyboard devices connected to the client side is encrypted before being transmitted to the server side for execution. The purpose of this mechanism is to expand the capabilities of the command transferring channel that was presented in [4], enriching the keyboard subsystem with encryption. More specifically, we examine how capturing from any input device on different end devices and on different operating systems is done, how public key encryption is applied and how commands are decrypted and executed at the target software of the server. In our work, we are considering only the confidentiality issues of the cryptographic module assuming that authenticity should be provided by the general architecture of the system or by a different module.

The rest of the paper is structured as follows: the next section describes the vision and goal of the Games at Large project. Section 3 describes the general architecture of the system and the architecture on each device (the end device and the server). Section 4 describes the encrypted command channel infrastructure which is the main scope of this manuscript while section 5 describes the general client-server infrastructure. The paper concludes with general remarks and future work that will be done.

II. Games at Large Project

Games at Large (Games@Large) being an Integrated Project (IP) intends to research, develop and implement a new architecture to provide users with a richer variety of entertainment experience in their houses, hotel rooms, cruise ships and Internet Cafés, incorporating unprecedented ubiquitous game-play. The project evolved from the home environment to other local Focus Areas (FA) regarding the benefits such FA may gain based on the unique technology approach of Games@Large. The Integrated Project includes activities of TV Multimedia and Gaming using Enhanced Media Extender, Local Processing and Storage Server(s), Handheld Devices and Local Wireless Network. Games@Large intends to enhance the existing Digital Living Network Alliance (DLNA) and the UPnP Forum standards by introducing the unique set of features required for running games over a local network, like all other media and content types (video, audio).

Games offer a leisure time activity for every member of the household – from avid gamers to kids, as well as allowing whole families to play together. Games offer also leisure time activity for guests in hotels and visitors in Internet Cafes. Games@Large offers ubiquitous accessibility for all members of the household on all desired entertainment devices. The project focuses on new innovative ideas such as multiple-game execution on the Games Gateway and delivery of graphics-rendering meta-data over the home network via low latency, low bandwidth Pre-Rendering Protocol to achieve low-cost implementation of ubiquitous game play throughout the house, while taking advantage of existing hardware, and providing multiple members of the family with the ability to play simultaneously.

Games@Large intends to enable the Games to diversify from dedicated appliances and a single corner of the house, to any place at home such as, the TV in the living room, the handheld device or any other device with the relevant screen, controls and connectivity. The project will also provide the required infrastructure for running games on the hotel guest room TV or on small screens for people sitting in Internet Cafés, cruse ships, trains or airplanes.

Some technological challenges of the Games@Large project are: Distributed computing and storage, video/image/graphics delivery with very low latency through a wired/wireless home network, adaptation of PC screen-images to TV screen and handheld devices, integration of wireless users' game control devices, translation of user ergonomics to different devices and form factors, research of new class of Media Extenders for games, enhancement of STBs to support video games, development of new methods for QoS linking consumer prospective with system measurements, enhancement of relevant industry standards for time critical multimedia content while maximizing users experience.

The project’s mission is to develop a new method for ubiquitous video games through unique technology to transfer graphical data while reducing latency and ensuring QoS in a cost-effective manner.

III. System Architecture

Fig. 1 depicts the general system architecture. As it is obvious, the system consists of two different “levels”. The first level includes all the servers that are used by the system, while the second level includes the connection of the different end devices of the system. The server side constitutes of multiple different servers that are assigned with the task of serving the games and require a high speed and stable communication between them while the second level is the interconnection of end devices with the server cluster in order to communicate and interact during the game play. While in the described architecture all the servers can communicate to one another, the end devices can actually interact only with the central Local Processing Server (LPS).

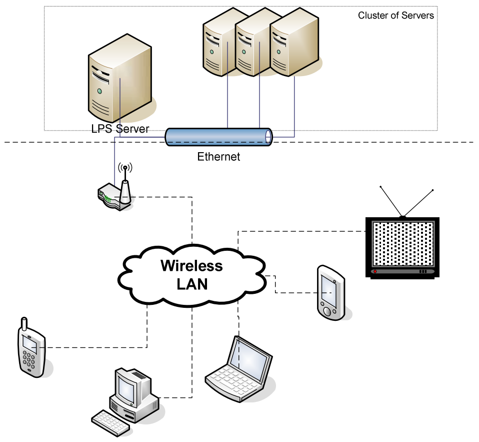

Fig. 1 General System Architecture

The server side of the system is assigned with various tasks the most important of which is that of executing the game and sending the corresponding scenario to the connected clients. The clients are constantly sending feedback to the server which describes the input commands that are to be executed to the game instance. Thus, the server should be able to have at least two communication channels with each client: one for sending the game frames or 3D commands (direct channel), and one for receiving the input from the clients for the game (reverse channel). An important aspect to notice is that the channel which, if hijacked, could jeopardize the system’s security, is the return channel since it contains not only the input commands that are for execution to the game instance, but also any other input from user. For instance, given the fact that the platform is targeted for commercial use, it is possible that the users will be required at some point to insert personal information, passwords, or even a credit card numbers. In this paper we will focalize on the encryption of the command communication channel and more specifically, on the encryption of keyboard input commands.

IV. Implementation Issues

As already noted, the encryption procedure is only needed for the keyboard commands that the client transmits. We will now briefly describe the initialization procedure for supporting RSA public key encryption both at the client and the server of the G@L environment, as well as the thereafter communication between the LPS server and the connected client.

A. *Startup phase*

When both the client and the server start, some local initializations take place (Fig. 2). Following, the client launches a connection request to the server which is advertised to the network neighborhood through the UPnP module. The server accepts the new client generating a unique RSA public-private key combination. Initially, through the persistent connection, the server transmits the modulus size in bits, the public exponent size in bits and the key pair size in bytes of the encrypted fields that follow.

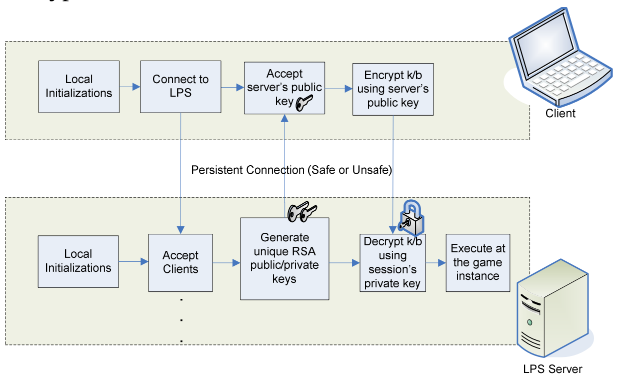
Local Initializations
Connect to LPS
Accept server's public key
Encrypt k/b using server's public key
Client
Persistent Connection (Safe or Unsafe)
Local Initializations
Accept Clients
Generate unique RSA public/private keys
Decrypt k/b using session's private key
Execute at the game instance
LPS Server

Fig. 2 Initialization of the encryption module

The public key is described by an RSA structure and its fields are transmitted sequentially to the client though the possibly unsafe channel. The client then accepts the structure's fields and re-generates the server's public key. From this point on, any keyboard commands are encrypted by the client using the server's public key and decrypted by the server providing thus the necessary security guarantee for the user-sensitive data.

B. *Transfer of encrypted keyboard input*

The idea that lies beneath the communication command channel architecture is depicted in Fig. 3 Each end device consists of many possible input devices for interacting with the server. As explained in [4], several steps need to take place in order to successfully deliver the input commands to the game process.

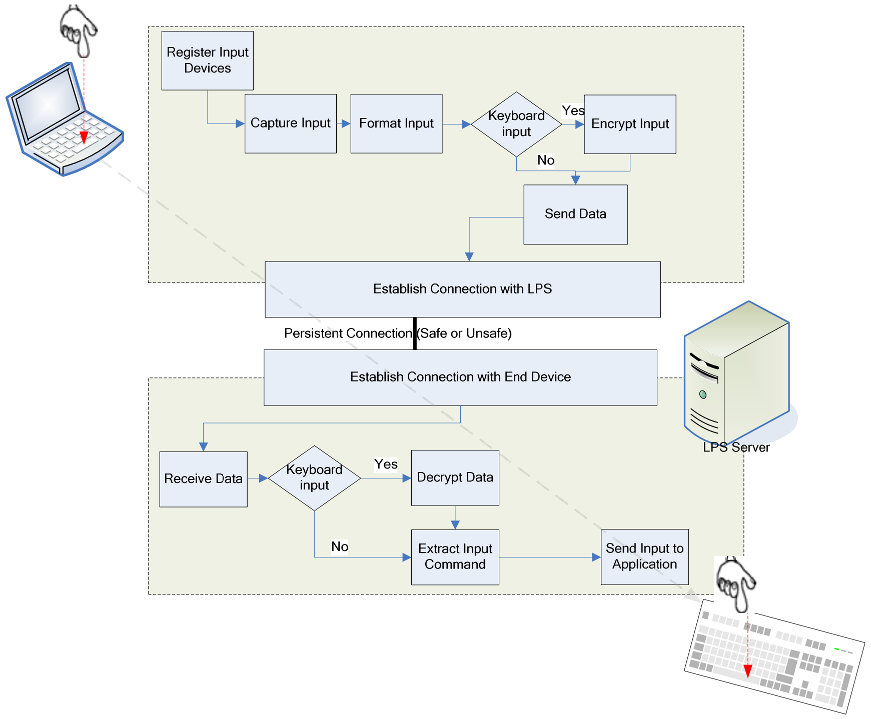

Fig. 3 Communication Command Channel

When the client program starts, it initiates the device discovery procedure, which may be offered either by a separate architectural module, e.g. the device discovery module which uses UPnP, or by a system call causing the discovery for input devices attached to the system. It is essential afterwards, that the results of the device discovery are registered in our program so that we are aware of the existing input devices marking out several other non-existing.

The next step of the procedure is to capture the input coming from the controllers. This is achieved by recording the key codes coming from the input devices. As explained in [4], input devices such as mice or keyboards are interrupt-driven while with joysticks or joy pads the polling method is used for reading. If the command that is to be transferred is originating from a keyboard device, the client uses the server's public key to encrypt the data after it has been suitably formatted adhering to a certain communication protocol. The encrypted message is transmitted to the server using an already open socket connection.

Once the encrypted message has arrived at the server side, the server decrypts it obtaining the initial keyboard commands that the client captured. If the received massage is not a keyboard one, the server bypasses the decryption stage, delivering the commands at the running game instance. The algorithm procedure of this step is presented in Fig. 4.

```
//-- Client Encrypts and Send Keyboard Data
int encrypt(string message) {
//pk_size is the public key size
  server.send_data(keyboard_type);
       //notify the server for
       //keyboard command that follows
  unsigned char *encrypted;
  int enc_size = RSA_public_encrypt(
            strlen(message)+1,
            (unsigned char *)message,
            encrypted, PUBLIC_KEY, PADDING);
  if (enc_size != pk_size)
  {
    Error("Ciphertext should match length of key");
    return(-1);
  }
//-- send encrypted data
  return server.send_data((char *)
            encrypted,enc_size);
}

//-- Server Receives and Dencrypts Keyboard Data
int dencrypt(string encrypted) {
  unsigned char *decrypted ;
  char temp [MSG_SIZE];
//-- receive encrypted data
  client.receive_data(temp,kp_size);
  memcpy((char *)encrypted, temp,kp_size*
                          sizeof(char));
  int decr_length = RSA_private_decrypt(kp_size,
  encrypted, decrypted, PRIVATE_KEY,PADDING);
  if(!decrypted){
    Error("Encryption failed");
    return -1;
  }
  Retrieve_vkey(decrypted);
}
```

Fig. 4 Encryption and Decryption of messages

V. Client - Server Infrastructure

The “gateway” of the servers is the Local Processing Server (LPS). The main goal of the LPS is to run multiple games simultaneously, whereas each game runs in its own game environment and is streamed to an end-device. The game environment is an isolated and encapsulated “sandbox,” providing the environment for game execution. The procedure, that makes the simultaneous running of multiple games possible, decouples the game execution from the game output, directed to display card/PC monitor, and all user-facing I/O, directed to the keyboard/mouse/HID. The LPS server also implements the encryption policy of the system by generating random RSA public/private key pairs for any newly connected clients and by decrypting the keyboard commands that come from the clients.

A wide variety of different clients can utilize the G@L environment: Laptops with Windows XP / Vista environment, Set-Top Boxes with either Linux or Windows CE and enhanced handheld devices with either Windows CE or a Linux version for small screen devices. Each client should implement the necessary RSA functions for the encryption module. For this cause, we are utilizing the OpenSSL RSA library which is available for the aforementioned platforms [10].

VI. General Remarks and Future Work

In this paper we have described the command execution channel of the Games at Large project, an IP project with the vision to research, develop and implement a new architecture to provide users with a richer variety of entertainment experience in their entire houses, hotel rooms, cruise ships and Internet Cafés, incorporating unprecedented ubiquitous game-play. As the system is implemented, more and more features are included on the release versions, such as modules that enable encryption of the commands and modules that utilize network specific characteristics in order to adapt on the possible network environment. Additionally, efforts are made towards the direction of creating software for every possible operating system in order to enable more end-devices to be connected to the Games at Large Environment.

References

[1] G. J. Simmons, “Symmetric and Asymmetric Encryption,” in ACM Computing Surveys (CSUR), vol. 11, no. 4, ACM Press New York, NY, USA 1979, pp. 305-330.

[2] P.S.L.M. Barreto, H.Y. Kim, B. Lynn, and M. Scott. “Efficient algorithms for pairing-based cryptosystems,” In Advances in Cryptology – CRYPTO 2002, volume 2442 of LNCS, pages 354–368. Springer-Verlag, 2002.

[3] S. Sattam, Al-Riyami and K. G. Paterson, “Certificateless Public Key Cryptography,” Lecture Notes in Computer Science, pp. 452 - 473, 2003

[4] C. Bouras, V. Poulopoulos, I. Sengounis and V. Tsogkas, “Networking Aspects for Gaming Systems,” Third International Conference on Internet and Web Applications (ICIW 2008), Athens, Greece, , 8 - 13 June 2008

[5] Games at Large project’s official website, http://www.gamesatlarge.eu

[6] World of Warcraft official website, www.worldofwarcraft.com

[7] Half Life official website, http://orange.half-life2.com/

[8] Second Life official website, http://www.secondlife.com

[9] Y. Tzruya, A. Shani, F. Bellotti, A. Jurgelionis, Games@Large - a new platform for ubiquitous gaming, BroadBand Europe 2006, Geneva, Switzerland, November 2006

[10] J. Viega, M. Messier, and P. Chandra, 2002. Network Security with OpenSSL, 1st Ed. O’Reilly, Cambridge, MA.