NTPsec
Sincronização de relógio com base num sistema remoto

Cláudio Esperança, Diogo Serra

2070030@my.ipleiria.pt, 2081008@my.ipleiria.pt

ESTG-IPLeiria, Leiria, Portugal

Resumo: Desenvolvimento de um sistema de sincronização de relógio seguro baseado no modelo computacional cliente-servidor[3] sob o protocolo de comunicação TCP[4]. A implementação em C# sob a plataforma .NET[5], traduziu-se em duas aplicações para o sistema operativo Windows onde, com recurso ao conjunto de camadas protocolares do TCP/IP[6], a aplicação cliente solicita informações sobre a data e hora do sistema à aplicação servidor. Para garantir a segurança da informação, a integridade, autenticidade e autorização nas trocas de dados foi implementado um protocolo de segurança adicional na camada aplicacional entre cada uma das aplicações.

Palavras-chave: Segurança, Informação, TCP, Sincronismo, Tempo, Confidencialidade, Autenticidade, Autorização, Integridade, Decifragem, Cifragem.

1 Introdução

A relação das pessoas com a tecnologia é simbiótica: estas investem em tecnologia e esta tenta tornar as suas vidas mais simples e fáceis. No entanto num mundo cada vez mais ligado, onde a tecnologia e os sistemas de informação assumem um papel cada vez mais essencial na vida das pessoas, a segurança torna-se assim numa necessidade para garantir a confidencialidade, integridade, autenticação e não repúdio da informação trocada entre os sistemas.

Apesar do conceito da segurança ser algo extremamente importante, o conceito de tempo é essencial para qualquer sistema informático. Deste conceito depende o sincronismo, utilizado internamente em qualquer sistema informático para gerir o seu funcionamento e externamente para trocar informações com outros sistemas[7]. Em sistemas críticos a funcionar sob redes IP[8] que dependam dos conceitos de tempo e sincronismo, o *Network Time Protocol*[9] (NTP) é um dos protocolos mais utilizados para sincronização de relógios, com tolerância a atrasos e a falhas, fornecendo uma aproximação à escala temporal *Coordinated Universal Time*[10] (UTC).

Existem vários exemplos de sistemas críticos que dependem destes conceitos. Por exemplo, o sincronismo é essencial para sistemas de desenvolvimento baseados em UNIX[11][12] onde o utilitário *make*[13] seja utilizado para a definição de instruções na execução de comandos de compilação. Esta ferramenta baseia-se na data de modificação associada a cada um dos ficheiros de um projeto para decidir quais os ficheiros que necessitam de ser recompilados. O sincronismo dos relógios[14] das máquinas de desenvolvimento torna-se assim essencial se for utilizado um repositório central partilhado para armazenamento dos ficheiros do projeto, pois se um dos relógios não estiver sincronizado, isto poderá implicar a recompilação de todo o projeto com as perdas de recursos associados.

O sincronismo é também essencial em sistemas de POS[15] de grandes superfícies onde os registos das transações sejam enviadas para um servidor central para processamento. Se os sistemas de venda não tiverem o seu relógio sincronizado com o servidor central, os registos poderão ter referências temporais inválidas o que pode comprometer a integridade das operações efetuadas.

Também no nosso caso de estudo o conceito de tempo é algo fundamental, como veremos com mais pormenor no capítulo 2 Assim, nos próximos capítulos começaremos por apresentar o problema, seguido da explicação da estratégia usada para a resolução do problema e consequente implementação. No final concluiremos o documento com a análise dos resultados e partilha das experiências adquiridas.

2 Verificação de período experimental em *trialware*

2.1 Identificação do problema

Em ambientes Microsoft Windows® é bastante comum a disponibilização de aplicações completas por um período de tempo durante o qual um utilizador pode experimentar todas as funcionalidades de uma aplicação (o chamado *trialware*[16] ou *demoware*). O período experimental inicia no momento em que o utilizador executa a aplicação pela primeira vez e, caso o utilizador não adquira a licença para utilizar o software, após término deste período a aplicação ativa o modo restrito onde apenas algumas ou nenhuma das funcionalidades estão disponíveis, no sentido de "incentivar" o utilizador a adquirir os direitos para utilizar o dito software. A verificação da utilização da aplicação no período experimental não pode ter apenas como base a data e hora locais, pois o utilizador pode alterar estas definições localmente no sentido de ludibriar a aplicação e estender assim o período experimental para além do permitido. Iremos propor uma solução para este problema através da deteção de diferenças temporais entre um servidor remoto e um cliente local, de forma segura, garantindo a confidencialidade, integridade, autenticação e não repúdio da informação trocada entre os sistemas.

2.2 Soluções atuais

Apesar do NTP possuir alguns mecanismos de segurança intrínsecos ao protocolo[17] [18], em alguns sistemas críticos pode existir a necessidade de utilizar mecanismos próprios que garantam a segurança da informação. É o caso do *trialware* onde é necessário garantir que a informação utilizada para a validação de um período experimental junto de um servidor central está isenta de ataques do tipo *men in the middle*[19] onde o próprio utilizador pode tentar assumir o papel do servidor para validar um acesso à aplicação que em condições normais não seria autorizado.

2.3 Solução proposta

Para resolver este problema sugere-se uma solução do tipo cliente-servidor, onde a aplicação cliente solicita a um servidor remoto devidamente certificado a sua data atual para utilizar como termo de comparação com a data local onde decorre o período experimental da aplicação.

Para garantir a confidencialidade das comunicações, as trocas de dados especificos da aplicação serão cifradas com recurso a um algoritmo de cifragem simétrico[20]. As definições a utilizar por este algoritmo (tais como a chave de encriptação[21] e o vetor de inicialização[22]) serão cifradas com recurso a um algoritmo assimétrico[40], com utilização da chave pública do destinatário da comunicação. A integridade das mensagens será garantida através do cálculo da *hash*[24] dos dados a enviar, que será

por sua vez assinada[25] com recurso à chave privada de cada remetente para garantir a autenticidade e o não-repúdio.

O funcionamento lógico da solução está representado na Ilustração 1 e será abordado com mais pormenor no capítulo seguinte.

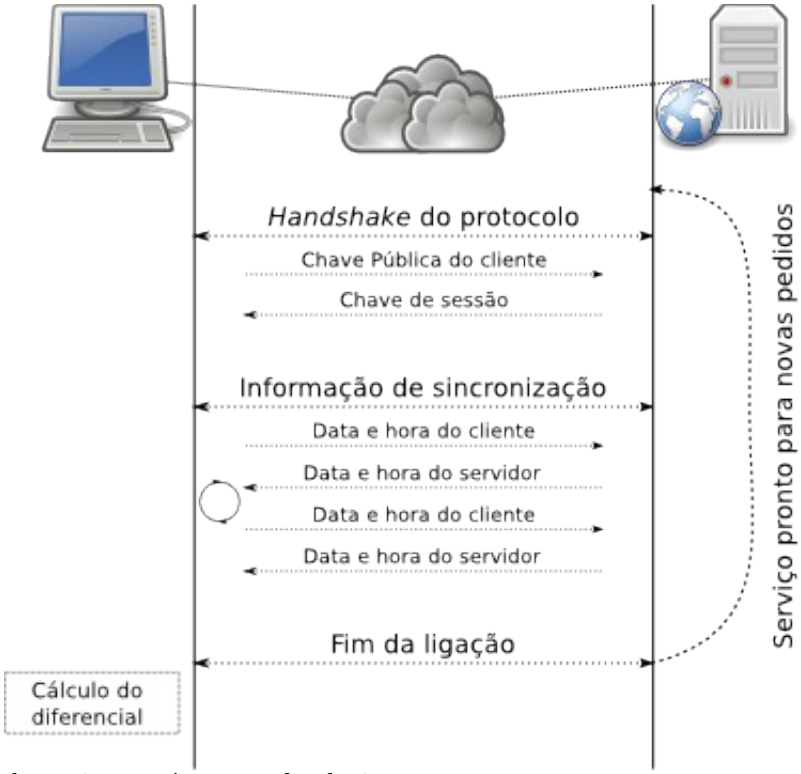

Ilustração 1: Funcionamento da solução

3 Mecanismo de comparação de relógios

3.1 Pré-requisitos

A chave pública do servidor é única e deverá ser distribuída com a aplicação cliente pois será com esta chave que se verificará a assinatura das mensagens enviadas. Este procedimento é necessário para garantir que a informação recebida pelo cliente teve origem num servidor fidedigno e que não foi alterada por terceiros durante a comunicação.

A chave pública da aplicação cliente pode ser gerada no momento da execução da aplicação dado que o servidor espera que esta seja enviada no inicio de processo de negociação, como está descrito no capítulo 3.2.

3.2 Handshake

O processo de negociação[26] inicia-se com o envio da chave pública do cliente para o servidor. Esta chave é necessária para que o servidor possa cifrar a chave de encriptação/sessão de forma a que apenas o cliente consiga decifrar esta chave utilizando a sua chave privada.

Para além de cifrar a chave de sessão, o servidor calcula a *hash* do texto cifrado para que o cliente possa verificar que não houve corrupção de dados durante o processo de comunicação.

Com a sua chave privada o servidor assina também a *hash* para que a aplicação cliente tenha como verificar se a mensagem partiu do servidor ou se foi enviada por terceiros.

O servidor envia então a chave de sessão cifrada e o resultado da assinatura da *hash* dos dados para o cliente. Este começa por calcular a *hash* dos dados que recebeu e, com recurso à chave pública do servidor, verifica se esta permite validar a *hash* assinada que foi recebida do servidor. Se o resultado da validação for positivo, a aplicação cliente utiliza a sua chave privada para decifrar a chave de sessão que deverá ser utilizada pelo cliente para cifrar os dados a enviar nas comunicações seguintes.

Este processo está resumido na Ilustração 2.

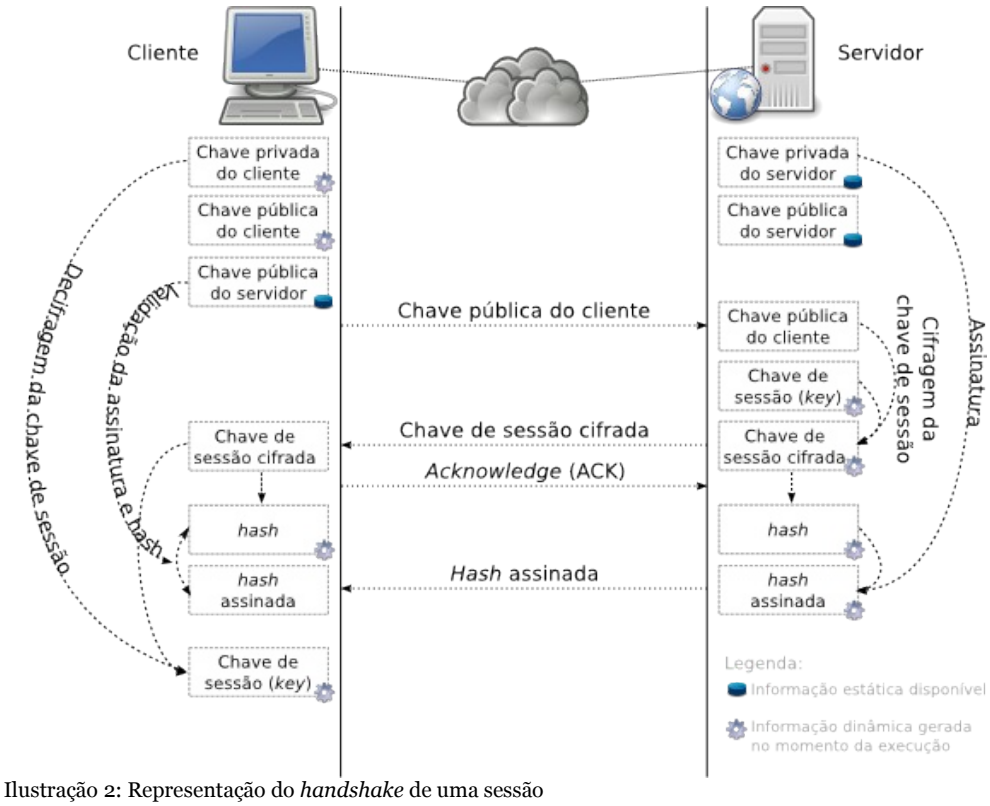

Ilustração 2: Representação do *handshake* de uma sessão

Após o processo de negociação, é tempo de avançar para o processo de sincronização.

3.3 Troca de informações de sincronização

Após a negociação inicial discutida no capítulo 3.2, é iniciada a troca de informação de sincronização, onde o cliente começa por registar localmente a sua data e hora atuais (L1), enviando essa informação para o servidor; o servidor responde ao

pedido com as suas informações de data e hora remotas; o cliente regista a data e hora do servidor (R1) e envia novamente a sua data e hora locais; o servidor devolve mais uma vez ao cliente a sua data e horas remotas que este regista localmente (R2) assim como a data e hora locais após a receção do pedido (L2).

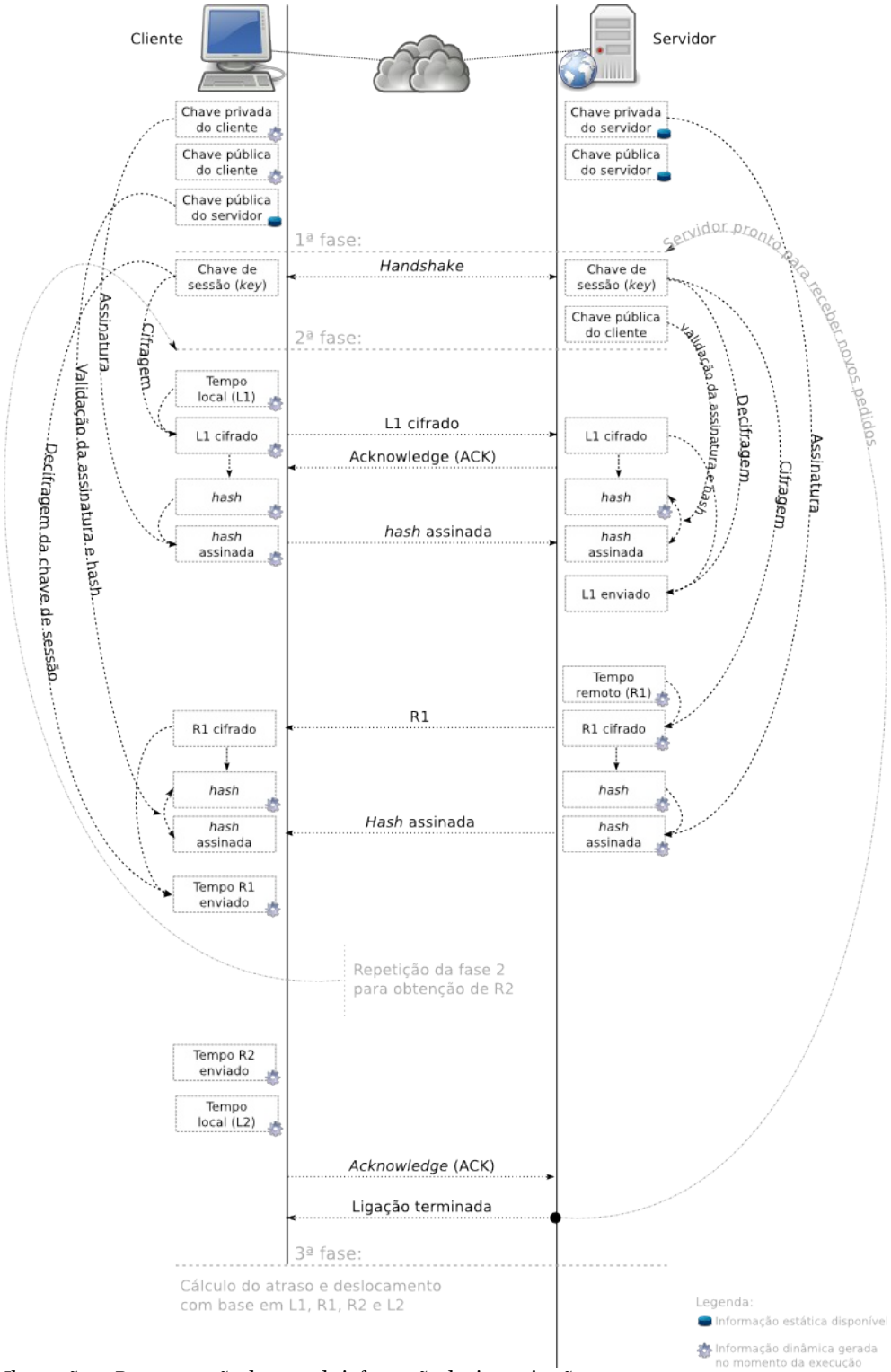

Ilustração 3: Representação da troca de informação de sincronização

Todos os dados são cifrados com a chave de sessão e é calculada a assinada a *hash* para esses dados, no sentido de garantir que todos os requisitos de segurança propostos são cumpridos.

3.4 Cálculo do atraso e do deslocamento

Após a troca de informações discutida no capítulo 3.3, são registados os seguintes valores:

- L1 – data e hora local do cliente no momento do primeiro pedido ao servidor
- R1 – data e hora remota do servidor no momento da receção do pedido do cliente
- R2 – data e hora remota do servidor no momento da receção do segundo pedido do cliente
- L2 – data e hora local após a receção da resposta do servidor ao segundo pedido

Com base nestas variáveis, o cálculo do atraso (A) é dado pela fórmula da Ilustração 4.

$$A=(L2-L1)-(R2-R1)$$

Ilustração 4: Cálculo do atraso

Por sua vez, o cálculo do deslocamento (D) pode ser calculado com a fórmula da Ilustração 5.

$$D=\frac{R1-L1+R2-L2}{2}$$

Ilustração 5: Cálculo do deslocamento

4 Implementação do NTPsec

No sentido de validar a solução proposta foi desenvolvido um projeto utilizando a tecnologia *framework*[27] .NET com recurso à linguagem de programação C#. Este projeto é constituído por duas aplicações (o NTPsec e o NTPsecClient), que pretendem replicar o funcionamento da solução num modelo cliente-servidor. Estas aplicações são do tipo *console application* e serão descritas com mais pormenor nos capítulos seguintes.

4.1 NTPsec

O NTPsec é a aplicação que assume o papel de servidor nesta implementação fornecendo informações temporais de forma segura a todos os clientes que as solicitem. Por omissão aguarda por ligações TCP na porta 13000 a partir de qualquer endereço de IP.

A aplicação começa por tentar carregar a chave privada e pública da aplicação servidor a partir do ficheiro XML (*serverPrivateKey.xml*) do sistema de ficheiros do sistema operativo; se esta operação for concluída com sucesso, as chaves serão utilizadas nas comunicações seguintes, onde se verifique a necessidade. Por outro lado, se os ficheiros com as chaves não existirem ou forem inválidos, a aplicação está preparada para, com base num par de chaves previamente definidas, criar estes ficheiros para utilizações futuras. Caso exista a recriação das chaves, é

necessário copiar a chave pública do servidor para a aplicação cliente para garantir que esta reconhece o servidor como uma entidade fidedigna no processo de comunicação.

A comunicação implementa o protocolo descrito no capítulo 3 sob a classe *ProtocolSI* disponibilizada nas aulas práticas e que serve de base a toda a comunicação. Para a cifragem simétrica foi também utilizada classe *SymmetricsSI* disponibilizada. Neste caso, dos métodos disponíveis, foram escolhidos os compatíveis com qualquer versão da *framework* .NET, para garantir a retro compatibilidade e portabilidade do código.

Nos mecanismos de cifragem simétrica dos dados, foi escolhido o algoritmo de cifragem *TripleDES*[29], com o modo de operação CBC[30][31] e com um modo de *padding*[33] PKCS7[32]. Estes parâmetros de cifragem estão definidos diretamente no código das aplicações e são comuns a ambos os nós. A chave de sessão e o vetor de inicialização são gerados pelo servidor no inicio da comunicação e enviados de forma cifrada (com recurso a algoritmos assimétricos) para o cliente.

Para cálculo das *hash* (para garantir a integridade da informação) e validação das assinaturas das mensagens, foi utilizado o algoritmo SHA1[34]. Para cifragem baseada no algoritmo chave pública foi escolhido o algoritmo RSA[35].

Apesar da aplicação servidor ser do tipo *single client*, foi implementado um mecanismo de *loop*[36] com o objetivo de permitir que a mesma aplicação servidor possa servir vários clientes em momentos diferentes. Do ponto de vista da implementação, a única função da aplicação servidor é fornecer *timestamps*[37] de forma segura, pelo que funcionalmente a aplicação funciona em modo de *loop* constante, onde no final de cada pedido é apenas devolvida da data e hora do servidor. Após a autenticação, cada pedido do cliente é respondido utilizando os parâmetros de cifragem negociados no *handshake*, sendo mantida a ligação cliente-servidor até ao momento em que o cliente envie um ACK, sinalizando o fecho da ligação.

A aplicação servidor utiliza também uma *thread*[38] secundária para disponibilização de um relógio que é apresentado na consola para referência visual. Pelo facto da posição de escrita deste elemento ser absoluta, foi necessário implementar um mecanismo de escrita para a consola que fosse *thread safe*[39] para evitar escritas em posições não desejáveis na consola (por exemplo se a *thread* principal tentar escrever texto para a consola enquanto a informação do relógio está a ser atualizada, poderão existir escritas na posição do relógio e não na posição objetivo). Para o efeito foram utilizados semáforos[40] de forma a garantir a atomicidade do processo de escrita para este recurso utilizando as funções *Write* e *WriteLine* disponibilizadas na aplicação.

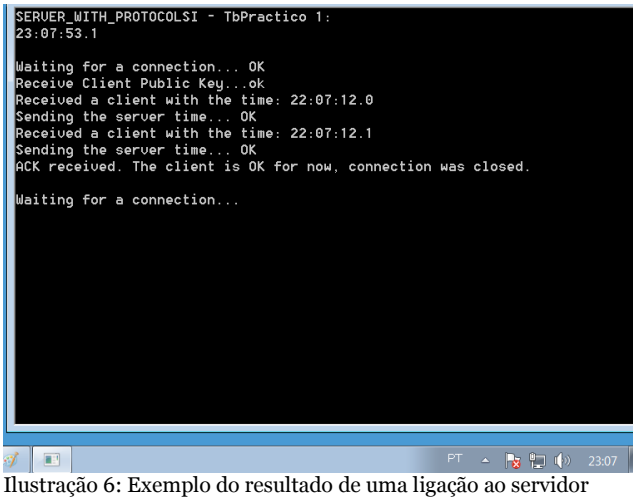

Ilustração 6: Exemplo do resultado de uma ligação ao servidor

4.2 NTPsecClient

Do ponto de vista da segurança, o funcionamento da aplicação cliente é semelhante ao da aplicação servidor. Os algoritmos e parâmetros de segurança são os mesmos, com a exceção da chave pública e privada do cliente que nesta implementação são geradas de modo dinâmico no momento da execução da aplicação. O ficheiro com a chave pública do servidor (serverPublicKey.xml) deve ser colocado junto ao executável da aplicação para certificar a autenticidade do servidor e, caso este ficheiro não exista, a aplicação cliente não comunica com o servidor. Numa implementação real para um sistema de produção, a chave pública seria incluída diretamente no código fonte da aplicação ou guardada em local seguro à qual não fosse permitido o acesso de escrita/modificação a terceiros, garantindo assim que só servidores devidamente credenciados pudessem ser utilizados para sincronização de informações de relógio.

À semelhança do que foi descrito no capítulo 3, é da responsabilidade do cliente o cálculo do atraso e deslocamento temporal da data e horas locais face à data e horas remotas. O cliente limita-se a solicitar *timestamps* ao servidor remoto para comparar com as suas medições locais e daí retirar as suas ilações.

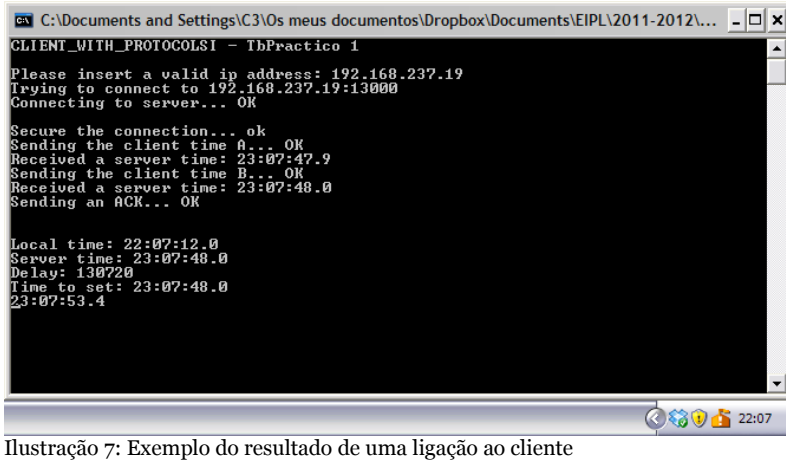

Ilustração 7: Exemplo do resultado de uma ligação ao cliente

5 Conclusão

Este estudo serviu para aprofundar conhecimentos na temática da segurança da informação. De um modo geral acreditamos que os objetivos foram cumpridos e a solução apresentada foi validada. Dos múltiplos sistemas e configurações testados foram obtidos os resultados pretendidos, onde se conseguiu de forma segura, a troca de informações de sincronização entre sistemas cliente e servidor.

As vantagens deste tipo de solução relacionam-se com os objetivos propostos, onde se conseguiu garantir uma comunicação segura, garantindo confidencialidade, integridade, não-repúdio e autorização. Como desvantagens temos o elevado número de comunicações entre os sistemas, necessárias para garantir a segurança, bem como o processamento extra necessários nos processos de cifragem e decifragem.

6 Referências

[1] NET Security and Cryptography, By Peter Thorsteinson, G. Gnana Arun Ganesh

[2] http://support.microsoft.com/kb/195724/en-us

[3] http://pt.wikipedia.org/wiki/Cliente-servidor

[4] http://pt.wikipedia.org/wiki/Transmission_Control_Protocol

[5] http://msdn.microsoft.com/en-us/library/zw4w595w.aspx

[6] http://pt.wikipedia.org/wiki/TCP/IP

[7] http://en.wikipedia.org/wiki/Synchronization_in_telecommunications

[8] http://en.wikipedia.org/wiki/Internet_Protocol

[9] http://en.wikipedia.org/wiki/Network_time_protocol

[10] http://en.wikipedia.org/wiki/UTC

[11] http://www.unix.org/what_is_unix.html

[12] http://en.wikipedia.org/wiki/Unix

[13] http://en.wikipedia.org/wiki/Make_(software)

[14] http://en.wikipedia.org/wiki/Clock_synchronization

[15] http://en.wikipedia.org/wiki/Point_of_sale

[16] http://en.wikipedia.org/wiki/Shareware

[17] http://www.faqs.org/rfcs/rfc1305.html

[18] http://www.eecis.udel.edu/~mills/database/rfc/rfc1305/rfc1305c.pdf

[19] http://en.wikipedia.org/wiki/Man-in-the-middle_attack

[20] http://en.wikipedia.org/wiki/Symmetric-key_algorithm

[21] http://en.wikipedia.org/wiki/Cryptographic_key

[22] http://en.wikipedia.org/wiki/Initialization_vector

[23] http://en.wikipedia.org/wiki/Asymmetric-key_cryptography

[24] http://en.wikipedia.org/wiki/Hash_function

[25] http://en.wikipedia.org/wiki/Digital_signature

[26] http://en.wikipedia.org/wiki/Handshaking

[27] http://en.wikipedia.org/wiki/Software_framework

[28] http://en.wikipedia.org/wiki/Console_application

[29] http://en.wikipedia.org/wiki/Triple_DES

[30] http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation#Cipher-block_chaining_.28CBC.29

[31] http://en.wikipedia.org/wiki/Padding_(cryptography)#CBC_mode

[32] http://en.wikipedia.org/wiki/PKCS

[33] http://en.wikipedia.org/wiki/Padding_(cryptography)

[34] http://en.wikipedia.org/wiki/SHA-1

[35] http://en.wikipedia.org/wiki/RSA_(algorithm)

[36] http://en.wikipedia.org/wiki/Loop_(computing)#Loops

[37] http://en.wikipedia.org/wiki/Timestamp

[38] http://en.wikipedia.org/wiki/Thread_(computer_science)

[39] http://en.wikipedia.org/wiki/Thread_safety

[40] http://en.wikipedia.org/wiki/Semaphore_(programming)