Documentação Completa: GCDLL.dll

Índice

Introdução

A biblioteca GCDLL.dll foi desenvolvida para oferecer um sistema de criptografia para comunicação cliente-servidor, especialmente projetada para jogos online ou aplicações distribuídas que exigem transmissão segura de dados. Esta biblioteca é projetada para trabalhar em conjunto com a interface Delphi definida em CryptLib.pas.

A biblioteca implementa funções de criptografia, descriptografia, geração de vetores de inicialização (IV) e finalização de pacotes para comunicação segura. A implementação foi feita em C++ para garantir alta performance e compatibilidade com a interface Delphi através do uso correto das convenções de chamada e tratamento de strings.

Descrição Funcional

A GCDLL.dll foi projetada seguindo rigorosamente os requisitos originais do autor, oferecendo as seguintes funcionalidades:

1. **Criptografia de Pacotes**: Transforma dados em formato seguro para transmissão.
2. **Descriptografia de Pacotes**: Restaura os dados originais a partir de pacotes criptografados.
3. **Geração de Vetores de Inicialização (IV)**: Cria sequências únicas para garantir que cada processo de criptografia seja diferente.
4. **Finalização de Pacotes**: Prepara pacotes criptografados para envio com verificação de integridade.

Estas funções foram implementadas para trabalhar de forma integrada com o sistema de manipulação de pacotes definido em CryptLib.pas.

Interface da API

A biblioteca exporta quatro funções principais que seguem a convenção de chamada `stdcall` para compatibilidade com Delphi:

```cpp
char* __stdcall _Encrypt(const char* data, const char* iv, unsigned char rnd);
char* __stdcall _Decrypt(const char* data, const char* iv);
char* __stdcall _GenerateIV(const char* ivHash, DWORD ivType);
char* __stdcall _ClearPacket(const char* data, const char* iv2);
```

Detalhamento das Funções

`_Encrypt`

- **Parâmetros**:
  - `data`: Pacote de dados a ser criptografado (AnsiString)
  - `iv`: Vetor de inicialização para personalizar a chave (AnsiString)
  - `rnd`: Valor aleatório para adicionar entropia à criptografia (Byte)
- **Retorno**: Pacote criptografado (AnsiString)
- **Comportamento**: Adiciona um byte aleatório como seed e aplica algoritmo de criptografia XOR personalizado com o IV.

`_Decrypt`

- **Parâmetros**:
  - `data`: Pacote criptografado (AnsiString)
  - `iv`: Vetor de inicialização usado na criptografia (AnsiString)
- **Retorno**: Pacote descriptografado (AnsiString)
- **Comportamento**: Remove o byte de seed e aplica o algoritmo inverso de criptografia usando o mesmo IV.

`_GenerateIV`

- **Parâmetros**:
  - `ivHash`: Hash base para geração do IV (AnsiString)
  - `ivType`: Tipo de algoritmo para geração do IV (DWORD)
- **Retorno**: Vetor de inicialização gerado (AnsiString)

- **Comportamento**: Gera um IV único baseado no hash fornecido e no tipo de algoritmo solicitado.

`_ClearPacket`

- **Parâmetros**:
  - `data`: Pacote já criptografado (AnsiString)
  - `iv2`: Segundo IV para verificação de integridade (AnsiString)
- **Retorno**: Pacote finalizado para envio (AnsiString)
- **Comportamento**: Adiciona verificação de integridade (checksum) baseada no conteúdo do pacote e no IV2.

Implementação Técnica

A implementação da biblioteca foi feita em C++ com as seguintes características:

Estrutura de Arquivos

- **GCDLL.cpp**: Implementação principal das funções da biblioteca
- **GCDLL.h**: Declarações de funções e protótipos
- **Makefile**: Script para compilação da biblioteca

Convenções e Compatibilidade

- Utilização da convenção `stdcall` para compatibilidade com Delphi
- Tratamento adequado de strings AnsiString
- Exportação explícita de funções para acesso externo

Gerenciamento de Memória

A biblioteca gerencia adequadamente a alocação e liberação de memória:

- Utiliza `new` para alocação de memória para strings retornadas
- As strings retornadas devem ser liberadas pelo chamador (a interface Delphi cuida disso)
- Evita vazamentos de memória durante as conversões entre formatos de string

Algoritmos de Criptografia

Algoritmo de Criptografia XOR

A biblioteca utiliza um algoritmo de criptografia baseado em XOR com as seguintes características:

1. **Chave Dinâmica**: Derivada a partir da combinação de:
   - Chave estática pré-definida (DEFAULT_KEY)
   - Vetor de inicialização (IV) fornecido

- Seed aleatório (rnd)

2. **Processo de Criptografia**:

```
Para cada byte de dados:
  dado_criptografado[i] = dado_original[i] XOR chave[i % tamanho_chave]
```

3. **Processo de Descriptografia**:

```
Para cada byte de dados criptografados:
  dado_original[i] = dado_criptografado[i] XOR chave[i % tamanho_chave]
```

4. **Seed Aleatório**: Adicionado no início do pacote para garantir que criptografias sucessivas do mesmo conteúdo gerem resultados diferentes.

Tratamento de Pacotes

Estrutura do Pacote

O tratamento de pacotes segue esta sequência:

1. **Pacote Original**: Dados originais a serem transmitidos
2. **Pacote após _Encrypt**:
   - Byte 0: Seed aleatório (rnd)
   - Bytes 1+: Dados criptografados
3. **Pacote após _ClearPacket**:
   - Bytes 0 a N-1: Pacote criptografado
   - Byte N: Checksum para verificação de integridade

Verificação de Integridade

O checksum é calculado da seguinte forma:

```
checksum = 0
Para cada byte i do pacote:
  checksum = checksum XOR pacote[i]
Para cada byte i do IV2:
  checksum = checksum XOR iv2[i]
```

Este checksum é adicionado ao final do pacote para permitir verificação de integridade na recepção.

Geração de Vetores de Inicialização

Os vetores de inicialização (IV) são cruciais para a segurança da criptografia. A biblioteca implementa três métodos de geração de IV:

Tipo 0: Derivação Simples do Hash

```
Para i de 0 até 15:
  iv[i] = ivHash[i % tamanho_ivHash] XOR DEFAULT_KEY[i % tamanho_DEFAULT_KEY]
```

Tipo 1: IV Baseado em Data/Hora

```
1. Obter data/hora atual do sistema
2. Formatar como bytes: ano, mês, dia, hora, minuto, segundo, milissegundo
3. Misturar com hash:
   Para i de 0 até 15:
     iv[i] = timestamp[i % 8] XOR ivHash[i % tamanho_ivHash]
```

Tipo 2: IV Aleatório

```
Para i de 0 até 15:
  iv[i] = numeroAleatorio(0-255) XOR ivHash[i % tamanho_ivHash]
```

Tipo Padrão (outros valores)

```
Para i de 0 até 15:
  iv[i] = (ivHash[i % tamanho_ivHash] + i) XOR DEFAULT_KEY[i % tamanho_DEFAULT_KEY]
```

Integração com CryptLib.pas

A biblioteca GCDLL.dll foi projetada para funcionar perfeitamente com a classe `TCryptLib` definida em CryptLib.pas. A integração funciona da seguinte forma:

Carregamento da Biblioteca

```pascal
constructor TCryptLib.Create;
begin
  hInst := LoadLibrary(PChar('GCDLL.dll'));
  if hInst = 0 then
    Exit;
  _Encrypt := GetProcAddress(hInst, '_Encrypt');
  if @_Encrypt = nil then
    Exit;
  _Decrypt := GetProcAddress(hInst, '_Decrypt');
  if @_Decrypt = nil then
    Exit;
  _GenerateIV := GetProcAddress(hInst, '_GenerateIV');
  if @_GenerateIV = nil then
    Exit;
  _ClearPacket := GetProcAddress(hInst, '_ClearPacket');
  if @_ClearPacket = nil then
    Exit;
end;
```

Fluxo de Operação

1. **Envio de Dados**:
   - Montar pacote original em `BIn`
   - Chamar `Encrypt` com IV e valor aleatório
   - Chamar `ClearPacket` para finalizar
   - Ajustar tamanho com `FixSize`
   - Enviar pacote
2. **Recepção de Dados**:
   - Receber pacote em `BOut`
   - Chamar `Decrypt` com IV correto
   - Processar dados descriptografados

Considerações de Segurança

A implementação atual da GCDLL.dll oferece um nível adequado de segurança para comunicações de jogos online ou aplicações similares, mas possui algumas limitações que devem ser consideradas:

Pontos Fortes

1. **Criptografia Personalizada**: Cada pacote utiliza uma chave diferente
2. **Verificação de Integridade**: Checksum para detectar modificações nos pacotes

3. **IVs Dinâmicos**: Múltiplos métodos de geração de IV

Limitações

1. **Algoritmo XOR**: Mais simples que algoritmos como AES ou RSA
2. **Chave Estática Base**: O DEFAULT_KEY é parte do código
3. **Sem Autenticação**: Não implementa assinaturas digitais

Recomendações

1. Alterar o DEFAULT_KEY antes da compilação para aplicações em produção
2. Considerar a adição de mecanismos de autenticação separados
3. Rotacionar IVs regularmente durante sessões longas

Exemplo de Uso

Aqui está um exemplo de como usar a biblioteca GCDLL.dll através da interface CryptLib.pas:

pascal

```pascal
var
  Crypt: TCryptLib;
  SendPacket: AnsiString;
begin
  // Inicializar a biblioteca
  Crypt := TCryptLib.Create;
  try
    // Configurar IVs
    Crypt.IV := 'BaseIVString12345';
    Crypt.IV2 := Crypt.GenerateIV(1); // Tipo 1: Baseado em data/hora

    // Montar pacote
    Crypt.BIn := #$01#$00#$00#$00#$00#$00; // Cabeçalho de 6 bytes
    Crypt.Write(Word(123));                // Escrever ID
    Crypt.Write(AnsiString('Hello'));      // Escrever dados

    // Criptografar pacote
    Crypt.Encrypt(Crypt.IV, Random(255)); // Usar valor aleatório

    // Finalizar pacote
    Crypt.ClearPacket;
    Crypt.FixSize;

    // Obter pacote para envio
    SendPacket := Crypt.BIn;

    // ... enviar pacote ...
  finally
    Crypt.Free;
  end;
end;
```

Compilação e Distribuição

Requisitos

- Compilador C++ com suporte a C++11 (GCC, Visual C++, etc.)
- Windows SDK ou MinGW para APIs Windows
- Make ou sistema de build equivalente

Compilação com GCC/MinGW

```
make -f Makefile
```

Compilação com Visual Studio

```
cl /EHsc /LD /O2 GCDLL.cpp /link /OUT:GCDLL.dll /DEF:GCDLL.def
```

Distribuição

Distribuir apenas o arquivo GCDLL.dll. Não é necessário distribuir os arquivos fonte ou de cabeçalho.

Instalação

Colocar o arquivo GCDLL.dll no mesmo diretório do executável principal ou em um diretório presente no PATH do sistema.

Solução de Problemas

Erros Comuns

1. **DLL não encontrada**:
   - Verificar se GCDLL.dll está no mesmo diretório do executável
   - Verificar permissões de arquivo
2. **Função não encontrada**:
   - Verificar se os nomes de função estão exatamente como esperado
   - Confirmar se a DLL foi compilada corretamente
3. **Corrupção de dados**:
   - Verificar se os IVs usados para criptografar e descriptografar são idênticos
   - Confirmar que o pacote não foi adulterado durante a transmissão
4. **Falha de Alocação de Memória**:
   - Verificar se o sistema tem memória disponível suficiente
   - Confirmar se os pacotes não excedem tamanhos razoáveis

Depuração

Para depurar problemas com a biblioteca:

1. Compilar uma versão de depuração com símbolos:

   ```
   g++ -g -Wall -std=c++11 -shared -o GCDLL.dll GCDLL.cpp
   ```

2. Usar ferramentas como Dependency Walker para verificar exportações
3. Implementar logging temporário para diagnóstico