ESCOLA POLITÉCNICA
DE PERNAMBUCO

d.s.c. UPE
Departamento de Sistemas e Computação

# ANÁLISE DE SEGURANÇA DE INFORMAÇÃO EM DISPOSITIVOS WUSB

## Trabalho de Conclusão de Curso

## Engenharia da Computação

**Felipe Zimmerle da Nóbrega Costa**
**Orientador: Prof. Carlos Alexandre Barros de Mello**

**Recife, junho de 2008**

UNIVERSIDADE
DE PERNAMBUCO

ESCOLA POLITÉCNICA
DE PERNAMBUCO

d.s.c. UPE
Departamento de Sistemas e Computação

# ANÁLISE DE SEGURANÇA DE INFORMAÇÃO EM DISPOSITIVOS WUSB

## Trabalho de Conclusão de Curso

## Engenharia da Computação

Este Projeto é apresentado como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

**Felipe Zimmerle da Nóbrega Costa**
**Orientador: Prof. Carlos Alexandre Barros de Mello**

**Recife, junho de 2008**

UNIVERSIDADE
DE PERNAMBUCO

**Felipe Zimmerle da Nóbrega Costa**

# ANÁLISE DE SEGURANÇA DE INFORMAÇÃO EM DISPOSITIVOS WUSB

# Resumo

A tecnologia USB, *Universal Serial Bus*, surgiu da necessidade de diminuir-se o número de protocolos de comunicação e de instituir-se um padrão de transmissão. Porém, a mobilidade é cada vez mais uma necessidade do usuário, e em decorrência desse fato, foi criada a tecnologia WUSB ou *Wireless* USB. O meio com fio não demanda uma grande preocupação com a proteção dos dados por ele trafegados. Contudo, o meio sem fio enfrenta mais obstáculos em relação à segurança, uma vez que, é mais propenso a sofrer interferências e ataques. Neste trabalho são efetuadas diversas análises sobre a segurança da informação no WUSB e seu protocolo de comunicação, além de melhorias nas implementações existentes. Também é realizado um estudo sobre criptografia, algoritmos de cifra e USB para embasar a compreensão das especificações WUSB. São analisados os *drivers* dos dispositivos HWA para aferir se estes encontram-se nos padrões de codificação do Kernel e no modelo de especificação WUSB. Também é realizada uma análise da implementação do *Cable Association* no sistema operacional Windows e no Linux. Ao longo do trabalho, também estão descritos os processos de teste e homologação das implementações de correção.

# Abstract

The USB, Universal Serial Bus, technology arose from the needs for reducing the number of communication protocols and to establish a transmission pattern. However, mobility is an increasingly need for the user, and it resulted in the establishment of the WUSB or Wireless USB technology. The wired USB transmission does not demand a great concern with data protection. However, wireless connections face more obstacles in relation to security, since it is more prone to suffer interference and attacks. This work does various analyses about information security in WUSB, its communication protocol, and improvements in existing deployments. Also it is presented an study on cryptography, encryption algorithms, and USB protocol in order to allow a complete comprehension of WUSB specifications. We make an analysis on device drivers of HWA devices to ascertain if they are found in the patterns of the kernel coding style and in the model of WUSB specification. It was also analyzed the implementation of the Cable Association in the Windows and Linux operating systems. The work describes the procedures for testing and approval of the necessary implementations.

# Sumário

# Índice de Figuras

# Índice de Tabelas

# Tabela de Símbolos e Siglas

(Dispostos por ordem de aparição no texto)

USB – *Universal Serial Bus*
WUSB – *Wireless* USB (USB sem-fio)
AES – *Advanced Encryption Standard*
DES – *Data Encryption Standard*
S-Box – *Substitution Box* (Caixa de substituição)
ECB – *Eletronic Code book*
CBC – *Cipher Block Chaining*
CFB – *Cipher Feedback*
OFB – *Output Feedback*
CM – *Counter Mode*
CCM – *Counter* com CBC-MAC
XOR – *Exclusive OR* (Ou exclusivo)
IV – *Initialization Vector* (Vetor de inicialização)
MAC – *Message Authentication Code* (Código de autenticação de mensagem)
USB-IF – *Universal Serial Bus Implementers Forum* (Fórum dos implementadores USB)
I/O – *Input/Output* (Entrada e saída)
IRPs – *I/O Request Packets* (Requisição de pacotes de I/O)
FIFO – *First in first out* (Primeiro a entrar é o primeiro a sair)
UWB – *Ultra Wide Band*
DWA – *Device Wire Adapter*
OOB – *Out of band* (Fora da banda)
HWA – *Host Wire Adapter*
PK – *Pair Keys* (Par de chaves)
CK – *Connection Key* (Chave de conexão)
CHID – *Connection Host Indentification* (Identificação de conexão de *host)*
CDID – *Connection Device Identification* (Identificação de conexão de dispositivo)
GTK – *Group Temporal Key*
CC – *Connection Context* (Contexto de conexão)
SFN – *Secure Frame Number* (Número do frame seguro)
TKID – *Temporal Key Identification* (Identificador de chave temporária)
MITM – *Men in the middle* (Home do meio)
WLP – *Wimedia Link Protocol*
IP – *Internet Protocol*
CBA – *Cable Association*
CBAF – *Cable Association Framework*
MMC – *Micro-scheduled Management Command*

# Agradecimentos

```
#include <linux/module.h>
#include <linux/config.h>
#include <linux/init.h>

static int __init mymodule_init(void)
{
        printk ("Agradeço em especial a minha família, meus pais, Flávia e Domingos por \n");
        printk ("todo o apoio e incentivo. A meus avós, em especial meu avô Ramon \n");
        printk ("Nóbrega, por todo carinho e por ter sido para mim, todos esses anos, um \n");
        printk ("grande exemplo.\n\n”);

        printk ("Ao meu sogro e sogra, Prof. Lee e Sonia, pelo incentivo e ajuda. \n");
        printk ("A minha cunhada, Sonmi, sempre disposta a nos levar pro mau caminho. \n\n");

        printk ("A todos os meus professores/amigos da faculdade especialmente a Carlos \n");
        printk ("Alexandre, que mesmo depois de me aturar durante dois anos seguidos \n");
        printk ("em Álgebra Aplicada, ainda assim, aceitou ser meu orientador. Obrigado. \n\n");

        printk ("A Ricardo Ulisses e a toda sua paciência por ter me aturado como dupla e \n");
        printk ("como colega de equipe em quase 100% dos trabalhos. \n\n");

        printk ("Agradeço a todos com quem trabalhei, que agüentaram meu mau humor e \n");
        printk ("minha cara de sono ☺. Especialmente, a Marco ‘Kiko’ Carnut por me passar \n");
        printk ("um pouco do seu conhecimento de criptografia e de l33t. Obrigado a clm, \n");
        printk ("ech, csm, jcpac, ^2, tcb, mabj e a todos os outros Tempesters. \n\n");

        printk ("Aos amigos do INdT, em especial ao meu chefinho Eduardo Rocha, por me \n");
        printk ("permitir trabalhar em projetos tão motivadores. A meu ex-chefinho ‘Frávio \n");
        printk ("José’, por tolerar meus atrasos e ausências. Han? O que? \n\n");

        printk ("A Inaky Perez e a Intel pela doação dos equipamentos utilizados durante \n");
        printk ("este trabalho. A Lizardo e a David Vrabel pela atenção e paciência na \n");
        printk ("revisão e nos comentários dos meus patches. \n\n");

        printk ("Agradeço a todos os que me ajudaram. Obrigado.\n\n");

        return 0;
}

static void __exit mymodule_exit(void)
{       printk ("Tks to Greg HK... I Love Greg. \n\n");      }

module_init(mymodule_init);
module_exit(mymodule_exit);

MODULE_LICENSE("Dual BSD/GPL");
MODULE_AUTHOR("Felipe Zimmerle <felipe@zimmerle.org>");
```

# Capítulo 1

# Introdução

Foi com o intuito de diminuir o número de protocolos de comunicação, instituindo-se um padrão de transmissão de dados entre periféricos e computador, que a tecnologia USB, *Universal Serial Bus*, foi criada. Hoje, a USB já é consagrada, pois possui inúmeras vantagens como: baixo custo, padrão robusto, redução do número de cabos e fácil utilização [1].

Cada vez mais a necessidade dos usuários se volta para a mobilidade e esta se alia à rapidez e agilidade nas comunicações. Atualmente, a interface de conexão eficiente e de alta velocidade USB já oferece as suas funcionalidades sem precisar dos fios. Um grupo de promotores da USB sem fio (também chamada de WUSB – Wireless USB) é que define as especificações que fornecem um padrão para a tecnologia. Esse grupo foi anunciado no Fórum de Desenvolvedores da Intel em 2004 e é formado por sete líderes no setor: Agere Systems [2], HP [3], Intel [4], Microsoft Corporation [5], NEC [6], Philips Semiconductors [7] e Samsung Eletronics [8].

A especificação da interface WUSB mantém a mesma utilização e arquitetura da USB com fio, definida por uma conexão host-dispositivo, de alta velocidade. O objetivo principal da especificação é fazer com que a WUSB funcione como um substituto do cabo. Para tal, é imprescindível que exista na WUSB o mesmo grau de segurança que na USB com fio. A nível de conexão entre dispositivos, a segurança garantirá que o dispositivo seja associado e autenticado para permitir seu funcionamento. Os níveis mais altos de segurança, como os que envolvem criptografia, por exemplo, devem ser implementados no nível de aplicação [9].

O meio com fio não necessita de uma grande preocupação com a proteção dos dados por ele trafegados. O meio sem fio, no entanto, é mais propenso a interferências e ataques, enfrentando mais dificuldades em relação à segurança. Fazer a equivalência da segurança provida por um meio físico em um meio sem-fio é uma tarefa que historicamente tem se provado desafiadora, principalmente em dispositivos que demandam uma alta velocidade de transmissão, onde uma menor latência ou o mínimo atraso pode prejudicar o bom funcionamento da comunicação e, por conseqüência, dos equipamentos. Tal desafio é somente aceito em benefício da facilidade e comodidade das transmissões sem-fio [10], [11].

Para obter uma comunicação segura deve haver garantia de autenticidade, integridade e confidencialidade dos dados transmitidos. Nesse contexto, a autenticação, que garante que ambas as partes em uma comunicação estão trocando informações com a entidade correta, é um fator importante uma vez que a comunicação no meio a ser protegido é compartilhada. Através da autenticação evita-se que um equipamento ou pessoa maliciosa se aproveite da arquitetura para

captura de dados ou ainda, que equipamentos com mau funcionamento - conhecido ou não - comprometam o sistema.

Garantir a comunicação segura entre os dispositivos é garantir a integridade dos dados bem como a autenticidade dos mesmos. Uma vez que os dados podem ser manipulados ainda em sua transmissão, isso pode gerar comportamentos inesperados que poderão acarretar até em uma negação de serviço do sistema. A integridade é a garantia de que os dados chegarão a seu destino de forma íntegra, ou seja, sem terem sofrido alterações, erros ou modificações indevidas [12], [13].

Não menos importante, a confidencialidade dos dados que estão sendo transmitidos no canal de comunicação é de extrema necessidade para proteção dos mesmos. É evidente que qualquer indivíduo mal intencionado pode, através de equipamentos adequados, capturar os dados que passam pelo canal, estes podem ser facilmente interpretados quando em texto claro. A confidencialidade é a garantia de que apenas as partes envolvidas em uma comunicação terão acesso às informações trafegadas [14], [15].

A comunicação com fio apresenta menos vulnerabilidades em relação à segurança das informações trafegadas devido à própria natureza do meio. Já, prover um meio de comunicação seguro para a transmissão de dados sem fio, cujas informações poderão ser críticas e em grandes volumes, respeitando o requisito inicial da praticidade com relação ao uso, demanda a utilização de criptografia aplicada diretamente em tal necessidade.

## 1.1 Objetivos

Este trabalho tem como objetivo o estudo dos conceitos de criptografia, incluindo alguns algoritmos de cifra, o estudo do funcionamento dos dispositivos WUSB, bem como suas respectivas especificações, a análise de segurança da informação nos dispositivos WUSB e melhorias na implementação do Linux para a segurança dos canais de comunicação propostos pelo padrão WUSB.

## 1.2 Organização do Trabalho

Este trabalho está organizado da seguinte forma:

- O Capítulo 2 apresenta os conceitos básicos de criptografia e exemplos de algoritmos de cifra.
- O Capítulo 3 descreve a arquitetura do padrão USB, apresentando suas topologias, além de explicar o seu modelo de comunicação.
- O Capítulo 4 define a tecnologia WUSB, apresenta suas topologias, modelo de comunicação, padrão criptográfico adotado e seus modelos de associação.
- O Capítulo 5 apresenta a análise realizada sobre o padrão de segurança do WUSB, incluindo uma criptoanálise da sua comunicação. Também é apresentada uma análise sobre os métodos de associação da especificação do padrão WUSB, além de uma análise do suporte ao WUSB no Linux. Neste Capítulo, são descritas as implementações realizadas para a correção dos problemas levantados.
- Por fim, o Capítulo 6 exibe as conclusões, contribuições, dificuldades encontradas e trabalhos futuros.

# Capítulo 2

# Conceitos Básicos

Este Capítulo apresenta alguns conceitos básicos, bem como exemplos de algoritmos criptográficos. Esses elementos são necessários para o bom entendimento do trabalho.

## 2.1 Definições Iniciais sobre Criptografia

Criptografia é um método de armazenar e transmitir dados de forma que, somente os destinatários possam processá-los e interpretá-los. É conhecida como a ciência de proteger informações por codificá-las em um formato não legível. A criptografia protege de forma eficaz o armazenamento e a transmissão de informações sensíveis em meios não seguros [16].

Existem alguns jargões da criptografia que devem ser explicados para a boa compreensão deste trabalho [17], [18], [19]:

- Cifrar – o mesmo que codificar, ou seja, converter informações sigilosas em algo sem sentido aparente.
- Decifrar – o mesmo que decodificar, ou seja, a função inversa de cifrar.
- Texto claro ou simples ou pleno – mensagem original, não cifrada.
- Texto cifrado – mensagem codificada, ininteligível para receptores não autorizados.
- Unidade de mensagem – subconjuntos do texto claro ou do texto cifrado.
- Função de cifragem – seja $S$ o conjunto de todas as possíveis unidades de mensagem em texto claro. Uma função de cifragem é uma função $f$ que aplicada a qualquer elemento do conjunto $S$, leve a algum elemento do conjunto $C$. Sendo $C$ o conjunto de todas as possíveis unidades de mensagem cifradas.
- Sistema criptográfico ou criptossistema – um conjunto de funções de cifragem e decifragem, além de outras funções que auxiliam no processo de cifra.
- Chave criptográfica – realizando uma analogia, uma chave criptográfica pode ser comparada a uma chave convencional e o algoritmo de criptografia a uma fechadura. Uma chave é utilizada na transformação de um texto claro em texto cifrado, ou vice-versa. Em um algoritmo de criptografia bem projetado, cifrar o mesmo texto com duas chaves diferentes, produzirá textos cifrados totalmente diferentes.
- Invasor – geralmente é alguém que tenta roubar informações. De forma geral, "invasor" é um termo que designa o indivíduo do qual as informações devem ser protegidas.
- Criptoanálise – Estudo sobre a quebra de sistemas criptográficos.

- Criptanalista – procura fraquezas que podem ajudar alguém a quebrar um algoritmo criptográfico mais rapidamente.

Além disso, a criptografia moderna exige que três propriedades sejam atendidas por qualquer criptossistema [13], [16]:

- Integridade: Garantia de que o dado que foi recebido, não sofreu nenhuma alteração durante a sua transmissão, ou seja, ele chegou no seu receptor de forma integra.
- Autenticação: Garantia do receptor sobre quem enviou a mensagem.
- Não-Repúdio: O emissor não pode negar o envio de uma mensagem.

Algoritmos de criptografia podem ser classificados como restritos, onde a segurança do algoritmo está baseada no próprio algoritmo; ou baseados em chave, onde a segurança está na proteção da chave [17]. Os algoritmos cuja segurança está na chave são os mais fortes. Tais algoritmos podem ser classificados em simétricos ou assimétricos como detalhado a seguir.

## 2.2 Criptografia Simétrica

Na criptografia de chave secreta, os processos de cifragem e de decifragem utilizam a mesma chave. Considere *Kc* a chave de cifragem e *Kd* a chave de decifragem, na criptografia de chave secreta: *Kc=Kd*, em razão disso, esse tipo de criptografia é também conhecida como simétrica [17].

Existem diversos algoritmos que fazem uso dos conceitos de criptografia simétrica, como, por exemplo, as cifras clássicas de Substituição e Transposição. Ambas possuem as principais técnicas utilizadas na construção dos algoritmos criptográficos simétricos modernos. São também utilizadas combinações dessas duas cifras em blocos de mensagens a serem cifrados ou decifrados. Um exemplo de tal algoritmo é o AES, *Advanced Encryption Standard* [20]. O AES surgiu para substituir o DES [21], *Data Encryption Standard*, algoritmo que virou padrão criptográfico a partir de 1977, desenvolvido pela IBM [22].

### 2.2.1 *Advanced Encryption Standard*

O AES, originário da cifra de Rijndael [17] - assim denominada pelos seus criadores, os belgas Joan Daemen e Vincent Rijmen - foi o algoritmo de cifra adotado como padrão pelo Governo Norte Americano desde novembro de 2001 e, sendo largamente utilizado em todo o mundo desde então. O AES destaca-se por não exigir muito poder computacional e por precisar de pouca memória, apresentando um bom desempenho tanto em *hardware* quanto em *software*.

O AES funciona com blocos de 128 bits e chaves de tamanhos que variam entre: 128, 192 e 256 bits. A mudança no tamanho das chaves e dos blocos apenas varia a quantidade de rodadas de permutações/substituições (operações base para o seu funcionamento). Para chaves de 128 bits, o algoritmo precisa de 10 iterações; 12 iterações são necessárias para chaves de 192 bits; e chaves de 256 bits exigem 14 iterações.

O funcionamento do AES é dividido em quatro operações principais, são elas:

- *SubByte* ou substituição de bytes.
- *ShiftRows* ou deslocamento de linhas.
- *MixColumns* ou permutação de colunas.
- *AddRoundKey* ou adição de chave de rodada.

Na operação *SubBytes* ocorre uma substituição não linear, onde cada byte do bloco é substituído pelo seu correspondente da tabela *S-Box* (*Substitution Box* ou Caixa de Substituição), conforme demonstrado na Figura 1a. A tabela *S-Box* consiste de um conjunto de valores pré-

definidos como descrito na Tabela 1. Observe, por exemplo, que o valor '0x9a' do byte do bloco corresponde ao valor 0xb8 na tabela *S-Box* (exemplo marcado em cinza na Tabela 1).

**Tabela 1.** Tabela S-Box usada pelo AES. As colunas determinam o nybble[1] menos significativo e as linhas o mais significativo [23].

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **A** | **b** | **c** | **d** | **E** | **f** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **00** | 63 | 7c | 77 | 7b | f2 | 6b | 6f | c5 | 30 | 01 | 67 | 2b | fe | d7 | Ab | 76 |
| **10** | ca | 82 | c9 | 7d | fa | 59 | 47 | f0 | ad | d4 | a2 | af | 9c | a4 | 72 | c0 |
| **20** | b7 | fd | 93 | 26 | 36 | 3f | f7 | cc | 34 | a5 | e5 | f1 | 71 | d8 | 31 | 15 |
| **30** | 04 | c7 | 23 | c3 | 18 | 96 | 05 | 9a | 07 | 12 | 80 | e2 | eb | 27 | b2 | 75 |
| **40** | 09 | 83 | 2c | 1a | 1b | 6e | 5a | a0 | 52 | 3b | d6 | b3 | 29 | e3 | 2f | 84 |
| **50** | 53 | d1 | 00 | ed | 20 | Fc | b1 | 5b | 6a | cb | be | 39 | 4a | 4c | 58 | cf |
| **60** | d0 | ef | aa | fb | 43 | 4d | 33 | 85 | 45 | f9 | 02 | 7f | 50 | 3c | 9f | a8 |
| **70** | 51 | a3 | 40 | 8f | 92 | 9d | 38 | f5 | bc | b6 | da | 21 | 10 | ff | f3 | d2 |
| **80** | cd | 0c | 13 | ec | 5f | 97 | 44 | 17 | c4 | a7 | 7e | 3d | 64 | 5d | 19 | 73 |
| **90** | 60 | 81 | 4f | dc | 22 | 2a | 90 | 88 | 46 | ee | **b8** | 14 | de | 5e | 0b | db |
| **a0** | e0 | 32 | 3a | 0a | 49 | 06 | 24 | 5c | c2 | d3 | ac | 62 | 91 | 95 | e4 | 79 |
| **b0** | e7 | c8 | 37 | 6d | 8d | d5 | 4e | a9 | 6c | 56 | f4 | ea | 65 | 7a | Ae | 08 |
| **c0** | ba | 78 | 25 | 2e | 1c | a6 | b4 | c6 | e8 | dd | 74 | 1f | 4b | bd | 8b | 8a |
| **d0** | 70 | 3e | b5 | 66 | 48 | 03 | f6 | 0e | 61 | 35 | 57 | b9 | 86 | c1 | 1d | 9e |
| **e0** | e1 | f8 | 98 | 11 | 69 | d9 | 8e | 94 | 9b | 1e | 87 | e9 | ce | 55 | 28 | df |
| **f0** | 8c | a1 | 89 | 0d | bf | e6 | 42 | 68 | 41 | 99 | 2d | 0f | B0 | 54 | Bb | 16 |

Na operação de *ShiftRows* cada linha sofre uma operação de deslocamento cíclico que se repete *n* vezes, onde *n* é o número da linha menos um. Essa operação está ilustrada na Figura 1b.

Em *MixColumns*, os quatro bytes de cada coluna são combinados através de uma transformação linear (Figura 1c).

Na etapa de *AddRoundKey*, cada coluna do dado é submetida a uma operação de XOR com a coluna correspondente da chave da rodada, conforme demonstrado na Figura 1d.

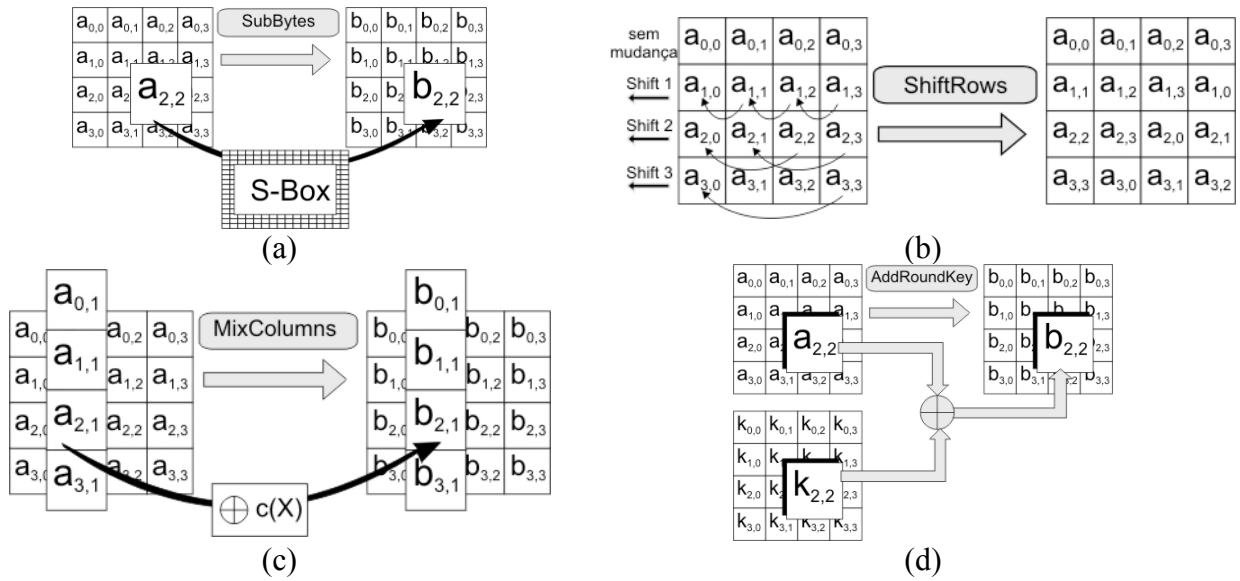



**Figura 1.** Operações do AES, (a) SubBytes. (b) ShiftRows. (c) MixColumns. (d) AddRoundKey.

[1] *Nybble* ou *nibble* é um termo que representa um conjunto de quatro de quatro bits, ou seja, a metade de um octeto. Um número em hexadecimal, representado por dois *nibbles*, um para cada digito [24].

Todas as quatro operações ocorrem em seqüência (Figura 2) e seu término resulta em um bloco cifrado.

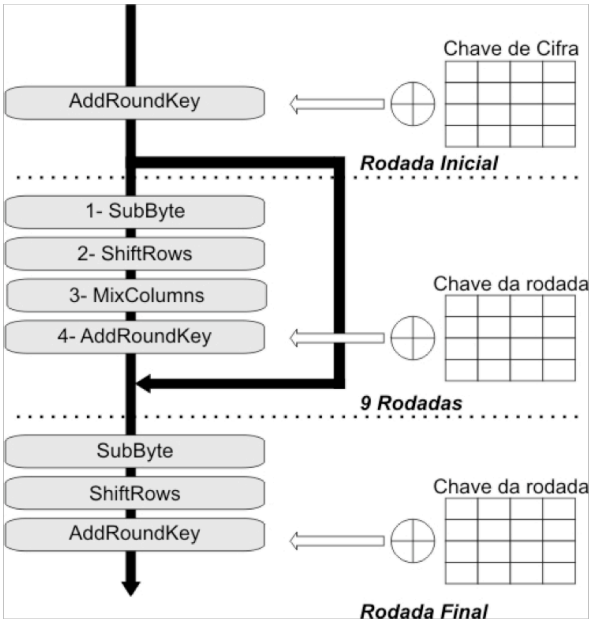



**Figura 2.** Etapas do AES.

Note que cada rodada faz uso de uma chave, esta depende diretamente do modo de operação escolhido para o funcionamento do algoritmo de cifra de blocos. Os modos de operação são detalhados a seguir.

### 2.2.2 Modos de Operação

Os algoritmos de criptografia de blocos, para realizar as suas operações, repartem os dados a serem cifrados em blocos menores. O modo de operação descreve um modelo utilizado pelo algoritmo de criptografia para que ele possa operar em cada bloco. Em geral, os modos de operação podem ser utilizados por qualquer algoritmo de cifra de blocos, possuindo extrema importância no processo de criptografia. A segurança final dos dados depende diretamente do modo de operação utilizado.

O AES, por exemplo, pode operar sob vários modos de operação. Os mais comuns são: *Electronic Code Book* (ECB), *Cipher Block Chaining* (CBC), *Cipher Feedback* (CFB), *Output Feedback* (OFB), *Counter Mode* (CM) e *Counter com CBC-MAC* (CCM). Neste trabalho, estão detalhados os modos de operação CBC, *Counter* e CCM, pois são importantes para os estudos realizados. Maiores detalhes sobre os outros modos podem ser encontrados em [18].

O modo de operação do CBC, ilustrado na Figura 3, consiste em criar uma cadeia entre os blocos, onde o resultado da cifra no bloco atual é usada juntamente com o texto claro a ser cifrado no bloco seguinte – a combinação do bloco cifrado com o texto claro do próximo bloco é realizada através de uma operação de Ou-Exclusivo (XOR – *eXclusive OR*). Nesse modo de operação, é necessária a utilização de um vetor de inicialização (IV, *Initialization Vector*). Este é utilizado por um criptossistema para fornecer certo grau de aleatoriedade ao processo de cifra. É necessária a criação de um bloco inicial que deve ser gerado a partir de números aleatórios/pseudo-aleatórios ou, em alguns casos, o próprio bloco a ser cifrado poderá ser utilizado como IV.

O IV, ou bloco falso, auxilia na cifra do primeiro bloco de dados. Na maioria dos casos, o IV não precisa ser secreto, entretanto ele nunca deve ser utilizado duas vezes com a mesma chave.

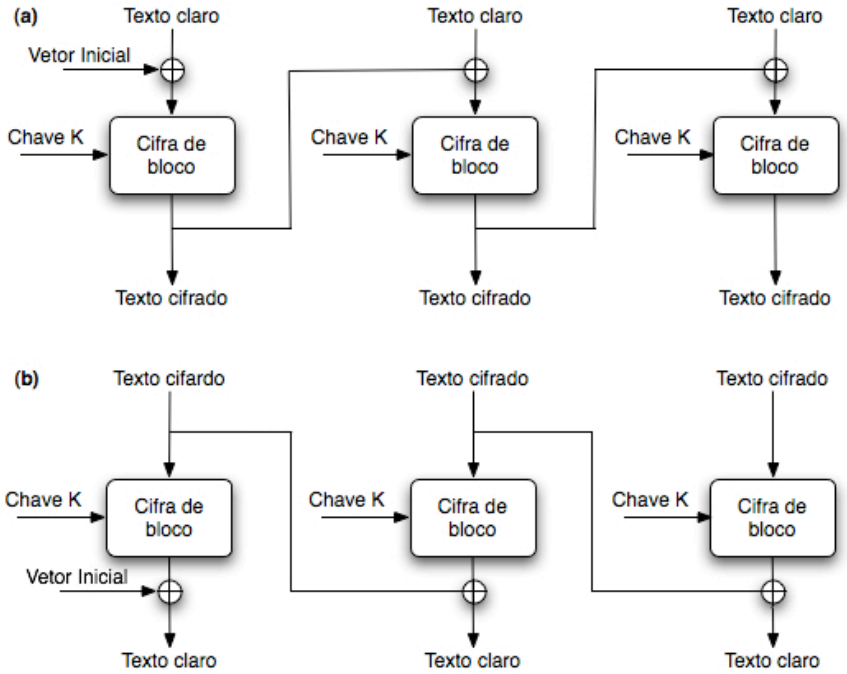



**Figura 3.** Modo de operação CBC, *Cipher Block Chaining*: (a) representa uma cifragem enquanto (b) representa um decifragem.

O modo *Counter*, ilustrado na Figura 4, é utilizado para cifrar a mensagem. Diferente do modo CBC, ele não depende do bloco anterior para realizar a criptografia, entretanto depende de um número incremental que é utilizado junto a cada bloco.

O modo CCM funciona como uma junção do modo CBC e *Counter*, sendo necessárias duas operações de cifragem para cada bloco. No CCM, também é utilizado o MAC, *Message Authentication Code*.

O MAC é uma informação utilizada para autenticar uma mensagem [25]. Um algoritmo gerador de MAC aceita como entrada uma chave secreta e um dado de tamanho arbitrário; como produto, recebe-se um MAC. Esse garante tanto a autenticidade quanto a integridade da mensagem, permitindo ao receptor detectar qualquer tipo de mudança ocorrida na mesma.

Assim como na criptografia simétrica, no MAC, a chave secreta é compartilhada entre o emissor o e receptor da mensagem. A Figura 5 apresenta um diagrama de operações do MAC.

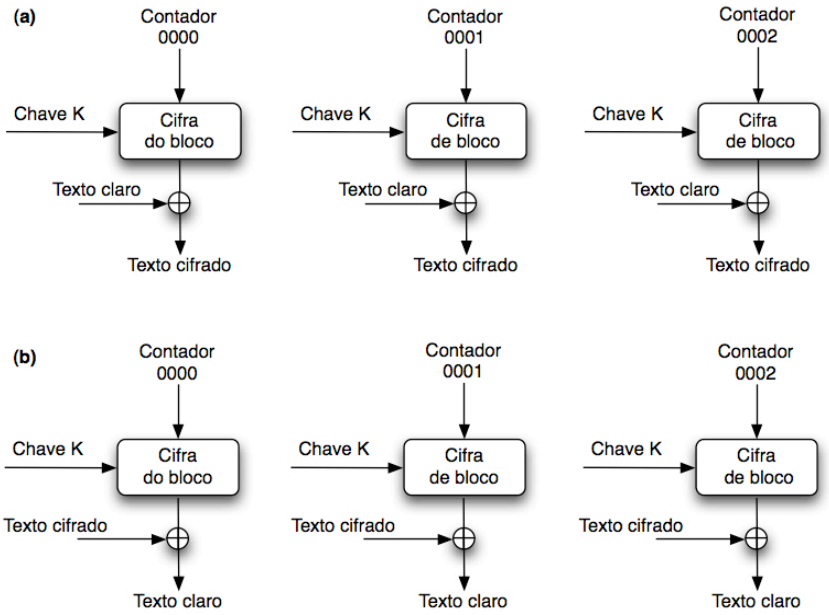



**Figura 4.** Diagrama de operação do modo *Counter*, sendo (a) o processo para cifrar e (b) o processo para decifrar.

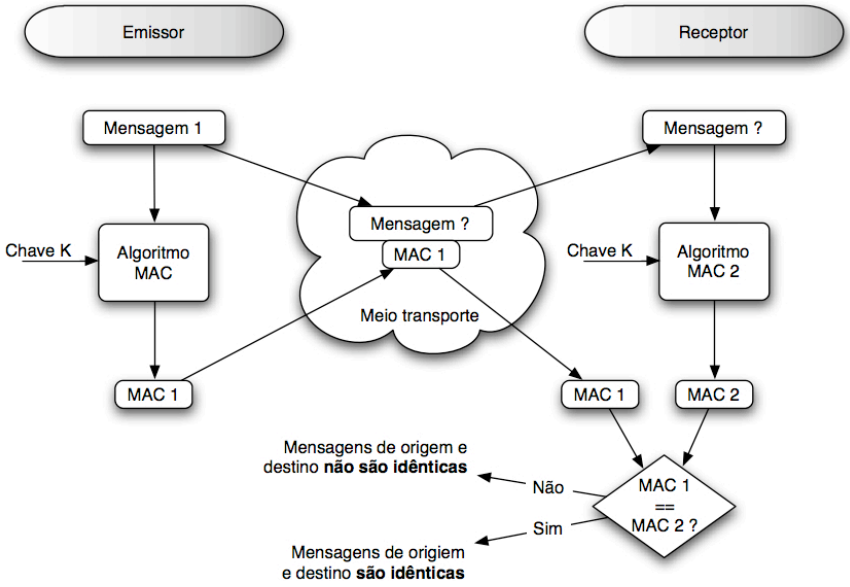



**Figura 5.** Diagrama de operação do MAC. Observe que a chave *K* é a mesma tanto para o receptor quanto para o emissor.

## 2.3 Criptografia Assimétrica

A criptografia assimétrica ou de chave pública, utiliza duas diferentes chaves, uma para cifragem e outra para decifragem, ou seja, $Kc \neq Kd$. Na criptografia assimétrica, cada um dos envolvidos é detentor de um par de chaves, sendo uma privada e uma pública, como pode ser observado na Figura 6. Como o próprio nome diz, a chave privada, ao contrário da pública, deve ser de conhecimento, só e somente só, do seu detentor. Essas chaves podem ser utilizadas de tal maneira a prover garantia de autenticidade, integridade e não repúdio de um dado.

Os algoritmos de criptografia assimétrica têm como característica principal o fato de serem computacionalmente muito mais caros que os algoritmos de criptografia simétrica,

entretanto destacam-se por não compartilhar uma chave secreta. Um dos mais populares algoritmos de criptografia de chave pública é o RSA [17] (criado por Rivest, Shamir e Adleman), publicado inicialmente em 1977 e bastante utilizado em transações eletrônicas, provendo segurança, inclusive, em conexões *web*. Uma ilustração do funcionamento do RSA pode ser vista na Figura 6.

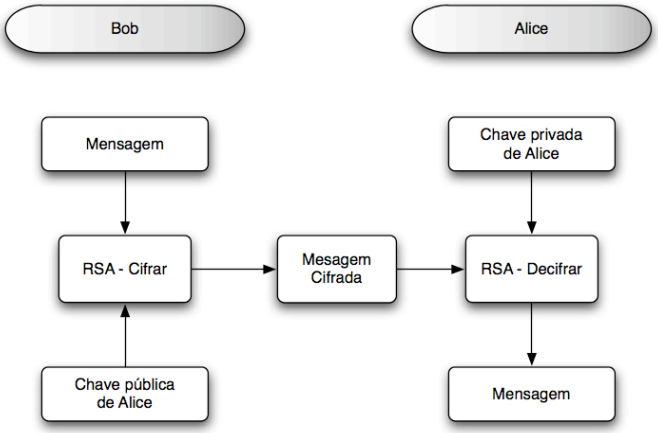



**Figura 6.** Exemplo do funcionamento do algoritmo de chave pública RSA.

## 2.4 Algoritmos de Hash

Hash Criptográfico, também conhecido como resumo de mensagem, é um algoritmo que gera uma saída de tamanho fixo. Essa saída é o resultado da manipulação de uma entrada de tamanho não definido. Na Figura 7, pode-se observar exemplos de Hash gerados a partir de frases [16], [17], [18].

A palavra *hash* tem sua origem na língua inglesa e significa desordem ou confusão. O algoritmo foi assim nomeado por gerar resultados pseudo-aleatórios, ou seja, que parecem não possuir uma ordem, apesar dela existir. O algoritmo de Hash Criptográfico tem por objetivo criar uma representação, ou resumo de uma mensagem, que pode ser processada por algoritmos de criptografia, cujo custo computacional é alto. Essa representação é usada para reduzir o custo, uma vez que o algoritmo pode ser aplicado a somente uma seqüência de caracteres de tamanho fixo – produto de um algoritmo de Hash, ao invés de, por exemplo, ser aplicada em todo o dado a ser assinado digitalmente.

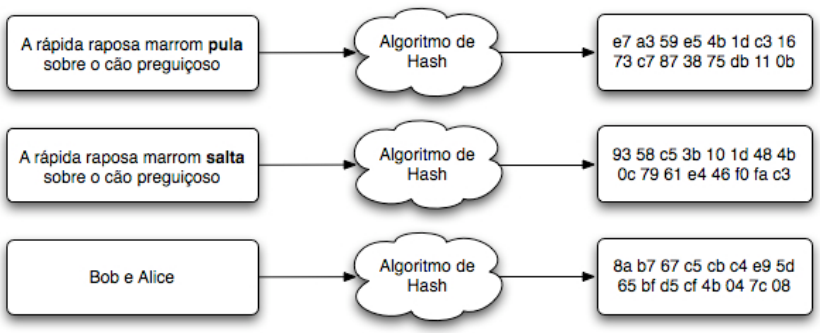



**Figura 7.** Exemplos de aplicações de um algoritmo de Hash.

Um bom algoritmo de Hash Criptográfico deve seguir um série de propriedades, dentre elas, podemos destacar:

- Dada a função de Hash *H* e um dado *d*, o cálculo de *H(d)* não deve ser computacionalmente caro;
- O resultado de *H(d)* tem um tamanho fixo, mesmo que o dado possua um tamanho arbitrário;
- A partir do resultado de *H(d)* não deve ser fácil encontrar *d*.

Uma vez que dados quaisquer são representados dentro de um conjunto finito, formado pelos possíveis resultados de um algoritmo de Hash, há a possibilidade de haver diferentes dados que são mapeados pelo algoritmo na mesma codificação hash. A isso se dá o nome de colisão e é inevitável que ela ocorra. Entretanto, a pseudo-aleatoriedade do resultado do algoritmo deve garantir que um resultado de hash específico não possa ser obtido através de manipulação de dados.

Atualmente, existem diversos algoritmos de Hash utilizados na criptografia, variando na sua forma de operar e no tamanho de suas saídas (Tabela 2). No entanto, apesar se suas características próprias, eles têm o mesmo objetivo. Devido ao crescente avanço computacional, alguns algoritmos caíram em desuso por tornarem-se suscetíveis à ataques de colisão.

**Tabela 2.** Comparativo entre diversos algoritmos de Hash.

| Algoritmo | Tamanho da saída (bits) | Tamanho do bloco de operações (bits) | Sujeito à ataques de colisão |
|---|---|---|---|
| HVAL | 256/224/192/160/128 | 1024 | Sim |
| MD2 | 128 | 128 | Sim |
| MD4 | 128 | 512 | Sim |
| MD5 | 128 | 512 | Sim |
| PANAMA | 256 | 256 | Sim |
| RIPEMD | 128 | 512 | Sim |
| RIPEMD-128/256 | 128/256 | 512 | Não |
| RIPEMD-160/320 | 160/320 | 512 | Não |
| SHA-0 | 160 | 512 | Sim |
| SHA-1 | 160 | 512 | Parcialmente |
| SHA-256/224 | 256/224 | 512 | Não |
| SHA-512/384 | 512/384 | 1024 | Não |
| Tiger(2)-192/160/128 | 192/160/128 | 512 | Não |
| WHIRLPOOL | 512 | 512 | Não |

# Capítulo *3*

# *Universal Serial Bus*

O padrão USB, *Universal Serial Bus,* foi projetado com o intuito de diminuir a quantidade de protocolos de comunicação e instituir-se um padrão de transmissão de dados entre periféricos e computador. O padrão se popularizou bastante pela sua simplicidade do uso, ganhando ainda mais usuários na sua versão 2.0 onde sua largura de banda passou a ser maior do que em sua versão anterior, atingindo 480 MB/s [26].

As duas características que mais contribuíram para a popularização do padrão USB foram: a facilidade de instalação de novos dispositivos e o fato da maioria dos dispositivos possuírem a propriedade *hot swap,* permitindo que eles possam ser ligados com o *host* - um computador ou vídeo *game*, por exemplo - sem a necessidade que o *host* seja previamente desligado. Uma outra característica importante no padrão é o fato de que a controladora USB provê energia suficiente para o funcionamento da a maioria dos dispositivos sem precisar de uma fonte de energia elétrica auxiliar.

Para garantir a interoperabilidade entre os diversos dispositivos que podem estar conectados num barramento, foi definida uma especificação. A autoria da especificação foi das mesmas empresas envolvidas no consórcio responsável pela criação do padrão do USB, o USB-IF (*USB Implementers Forum*). Os dispositivos que seguem tais especificações recebem um selo de compatibilidade, como ilustrado na Figura 8. Novas versões do padrão USB e suas respectivas especificações, também, são definidas pelo USB-IF [27].


(a) (b)

**Figura 8.** Logos colocados em dispositivos USB que seguem o padrão espeficado pelo USB-IF. A imagem (a) representa os dispositivos 2.0 enquanto a (b) representa os dispositivos nas versões anteriores à 2.0.

O padrão USB não somente especifica o modo de comunicação entre o dispositivo e o *host*, como também especifica o modo de funcionamento de algumas classes de dispositivos, como: teclados, *mouses*, adaptadores *bluetooth*, entre outros, que podem ser suportados pelo sistema operacional sem a necessidade de um *driver* específico.

## 3.1 Arquitetura do padrão USB

Os sistemas operacionais provêm uma interface – também respeitando os padrões do USB-IF – que abstrai do desenvolvedor problemas relacionados à forma da comunicação, tornando mais simples a programação de *drivers* para os dispositivos. A comunicação USB depende de três elementos básicos, são eles:

- USB *Interconnect* – Provê o meio de comunicação e transporte de dados entre o dispositivo e o *host*.
- USB *devices* – São os dispositivos utilizados pelo usuário, por exemplo, HUBs, *joysticks*, *mouses* e teclados.
- USB *host* – É o responsável pela gerência de operações no barramento. Permite uma boa comunicação e fornece interfaces de acesso às informações para um meio final. Esse meio final pode ser um computador, um vídeo game, um *media center* ou até mesmo um som automotivo.

### 3.1.1 Topologia Física

Os dispositivos USB são conectados fisicamente ao *host* e utilizam uma topologia estrela[2], possuindo, assim, uma hierarquia que permite, por exemplo, o uso de potência elétrica externa através de um dispositivo especial chamado de HUB. A função do HUB é aumentar o número de portas de entrada para o *host*. Cada *host* USB tem um *root* HUB (interno) que faz a ligação entre o *host* e o dispositivo. Não existe conexão direta de um dispositivo com o *host*, como pode ser observado na Figura 9.

HUBs especiais fazem parte de dispositivos chamados de "Dispositivos compostos" e têm o papel de ligar duas ou mais classes pertencentes fisicamente ao mesmo dispositivo.

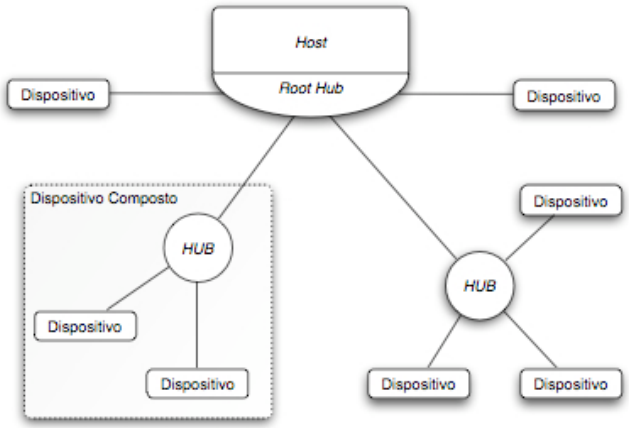



**Figura 9.** Topologia física da ponte USB [26].

### 3.1.2 Topologia Lógica

[2] Rede estrela: Diz-se topologia estrela quando um dispositivo se conecta a outro apenas através de um equipamento central, concentrador, sem nenhuma ligação direta entre os dispositivo [28].

Na topologia lógica, os dispositivos USB são conectados diretamente ao *host*, sem nenhuma distinção quanto à hierarquia. Também na perspectiva lógica, não são representados os HUBS, sendo esses transparentes na comunicação entre o dispositivo e o *host*. A Figura 10 representa a topologia lógica de uma ponte USB.

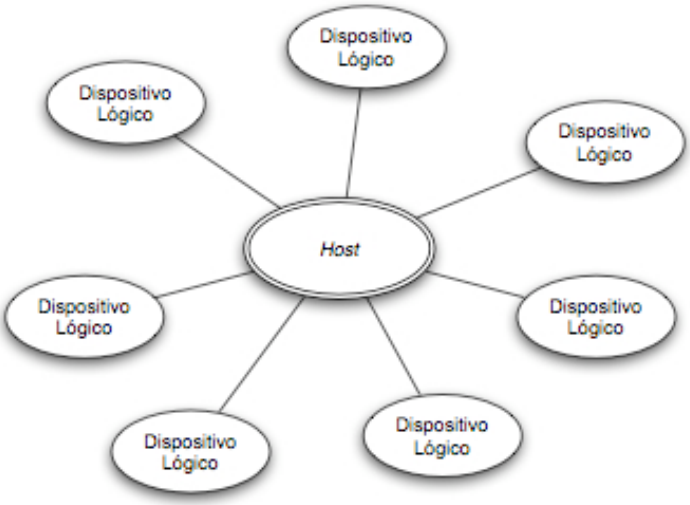



**Figura 10.** Topologia Lógica da ponte USB [26].

## 3.2 Modelo de Comunicação

A comunicação USB entre o *host* e o dispositivo é realizada através de canais de comunicação chamados de *pipes*, cada um desses *pipes* tem *buffers*[3] próprios.

Um canal de controle é utilizado para gerência de dispositivos ligados ao barramento. Esse canal de controle tem um *endpoint* com identificador 0 (zero) o qual pode trocar informações nas duas direções: entrada e saída. Ele também é utilizado para configuração do dispositivo (assim que conectado ao *host)* e na troca de mensagens de *status*.

Um *endpoint* representa o término de uma comunicação entre um *host* e um dispositivo. Cada dispositivo lógico USB é composto por um conjunto de *endpoints* independentes. Juntando o *endpoint* com o endereço do dispositivo – que é dado pelo *host* no momento que o mesmo é conectado ao barramento – e a direção de uma mensagem, é possível identificar, unicamente, uma mensagem no barramento. Na Figura 11 pode ser observada a presença desses identificadores na assinatura da função que envia mensagens de controle no Kernel do Linux[4].

[3] *Buffer* é uma região de memória utilizada temporariamente por um dado que será movido para outro lugar. Um dado fica num *buffer* esperando em uma fila, por exemplo.

[4] Kernel ou cerne do Linux, e o núcleo do sistema operacional. Ele é responsável por prover uma interface entre os *softwares* e *hardwares* de um computador [29].

```
int usb_control_msg(struct usb_device *dev, unsigned int pipe, __u8 request, __u8 requesttype,
                    __u16 value, __u16 index, void *data, __u16 size, int timeout)
```

**dev**: Ponteiro para a estrutura que define o dispositivo USB no kernel.
**pipe**: O *endpoint* para o qual a comunicação será estabelecida.
**request**: Requisição.
**requesttype**: Tipo de requisição.
**value**: Valor da menssagem.
**index**: Valor de índice da mensagem.
**data**: Ponteiro para estrutura que contém os dados ou que conterá.
**size**: Tamanho dos dados recebidos ou enviados.
**timeout**: Mili segundos de espera para completar a transição.

**Figura 11.** Assinatura da função de envio de mensagem de controle para um dispositivo USB, definida no *kernel* do Linux, versão 2.6.17 no arquivo drivers/USB/core/message.c e encontrada na linha 135.

A direção da mensagem num barramento pode ser de *input* (do dispositivo para o *host*) ou de *output* (do *host* para o dispositivo). Cada troca de menssagem com um *endpoint* tem características que determinam o tipo de transferência. Essas transferências são compostas de:

- Freqüência/Latência de acesso;
- Largura de banda requerida;
- Identificador;
- Requerimentos para tratamento de erro;
- Tamanho máximo do pacote que pode ser tratado pelo *endpoint*;
- Tipo de transferência que poderá ser feita;
- A direção dos dados.

Os *pipes* de comunicação apresentam-se de duas formas:

- *Stream* - Não existe uma estrutura que pré-defina a forma dos dados.
- Mensagem - Existe alguma definição do tipo de dados que passa no canal.

As mensagens enviadas pelo barramento não são interpretadas enquanto estão no canal de comunicação, mesmo quando são mensagens de controle onde a sua forma é conhecida.

A comunicação *Stream* é iniciada com o requerimento de transferência de dados via *I/O Request Packets* (IRPs) para um *pipe* que ao ser concluída gera uma notificação. Caso não haja nenhuma IRPs pendentes ou em progresso, o estado do canal será *idle* – em espera - até que uma requisição ocorra.

Os *pipes* do tipo *Stream* são geralmente uniderecionais e respeitam a ordem seqüencial FIFO (*first in first out)* de transferência de dados. A Figura 12 representa a comunicação entre um dispositivo e um *host*.

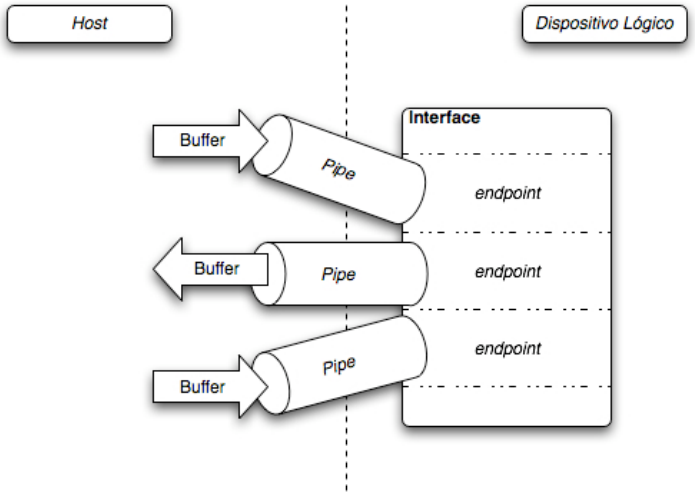



**Figura 12.** Comunicação entre um *host* e um dispositivo lógico [26].

Os *pipes* do tipo mensagem funcionam em um modelo diferente do *Stream*, onde uma mensagem é enviada e sua confirmação de recebimento é enviada em seguida (de maneira síncrona).

# Capítulo 4

# *Wireless* USB

Neste Capítulo, abordamos detalhes sobre WUSB. Detalhes sobre a topologia, modelo de comunicação adotado bem com os modos de associação entre dispositivos e *hosts*.

## 4.1 Sobre WUSB

Unindo as características do padrão USB com a facilidade e a praticidade da transmissão de dados sem-fio, foi criado o padrão *Wireless* USB ou, comumente chamado, USB sem-fio. Tal padrão foi definido pelo mesmo fórum mantenedor do padrão USB e consiste no uso de redes UWB, *Ultra Wide Band*, para transmissão de dados com baixa latência e alta largura de banda.

Mantendo a compatibilidade com os dispositivos USB, o padrão *Wireless* USB mantém as mesmas características básicas do USB. Algumas outras propriedades foram definidas com o propósito de atender às características relacionadas à dispositivos sem-fio, como o caso da autorização de conexão ao barramento, que deve ser realizada antes da conexão de qualquer dispositivo.

A segurança da transmissão dos dados - entre o *host* e o dispositivo - passou a ser descrita na especificação WUSB. Essa preocupação não existia no padrão USB, pois o próprio meio de comunicação (fio) utilizado garantia a segurança desejada, uma vez que os fios dos dispositivos USB são curtos o suficiente para serem observados pelo seu utilizador.

Assim como os dispositivos USB com-fio, os dispositivos USB sem-fio também ganharam um logo (Figura 13) que indica sua compatibilidade com o padrão WUSB. O logo agora representa não só uma boa comunicação, mas uma comunicação segura.


**Figura 13.** Logo que representa a garantia de compatibilidade dos dispositivos *Wireless* USB com o padrão estabelecido pela USB-IF.

Os conceitos de *host* e dispositivo, vistos no padrão USB, são respeitados no padrão WUSB, destacando, porém, que nesse último, o meio de propagação das informações não é mais um fio.

Dado que não existe mais a necessidade de um dispositivo HUB, uma vez que as entradas físicas não são mais necessárias, esse dispositivo passou a ser desconsiderado pelo padrão WUSB. Contudo, outro dispositivo especial passou a existir. Este é chamado de DWA, *Device Wire Adaptor*, e tem a função de permitir a conexão entre dispositivos com fio no barramento sem-fio, fornecendo não só a conexão no barramento mais também a potência elétrica necessária para o seu funcionamento.

### 4.1.1 Topologia Física

No padrão WUSB não há conexões físicas dos dispositivos com o *host*. Entretanto, os dispositivos que estão dispostos num raio de até 10 metros distância são capazes de estabelecer comunicação com o *host*. A Figura 14 exemplifica a topologia física de uma rede de dispositivos WUSB, incluindo o uso de um dispositivo DWA.

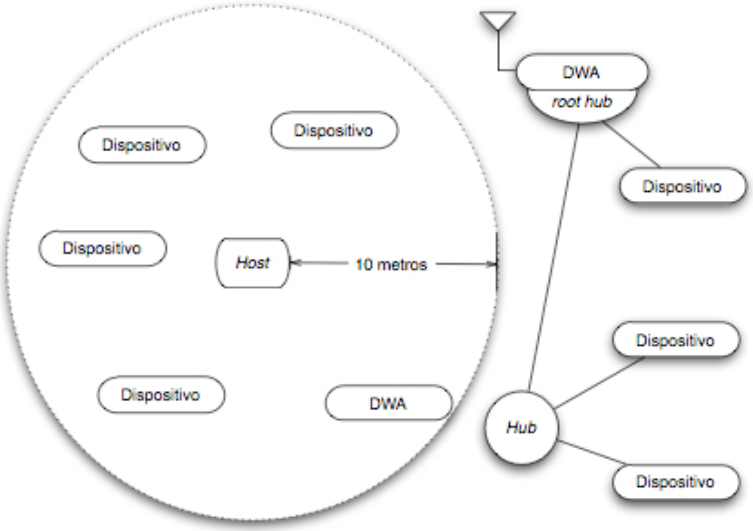



**Figura 14.** Exemplo de dispositivos conectados a uma rede WUSB [30].

Existem alguns dispositivos que para troca inicial de chaves – utilizadas para estabelecer uma comunicação segura – necessitam ser ligados fisicamente a um barramento USB, fazendo a troca de chaves *Out of Band* (OOB) ou fora da banda.

### 4.1.2 Topologia Lógica

Utiliza-se como topologia um modelo que se assemelha ao modelo estrela cujo centro é o *host* e os dispositivo são as pontas finais. Dado que os dispositivos WUSB não dependem de portas físicas, a limitação de conexão de dispositivos depende diretamente da quantidade de dispositivos sem-fio suportada pelo protocolo, sendo esse limite de 127 dispositivos. A Figura 15 ilustra o modelo da topologia WUSB.

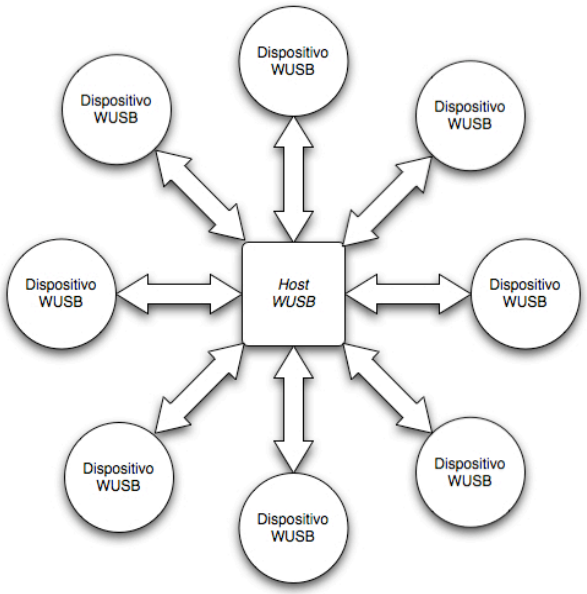



**Figura 15.** Topologia WUSB [30].

Também chamado de HWA, *Host Wire Adapter,* o *host* UWB pode ser adicionado no sistema através do barramento USB (necessitando de um barramento USB 2.0).

### 4.1.3 Modelo de Conexão

Dado que não existe uma conexão física entre o *host* e o dispositivo, uma série de premissas devem que ser cumpridas para que se estabeleça uma conexão lógica. Para essa comunicação, o dispositivo WUSB assume vários estados que são relevantes para o *host*. O objetivo de seguir esses critérios é não só a criação de uma conexão, mas sim uma conexão segura antes que dados críticos passem no canal.

Afirmar que uma conexão é segura não é dizer que é impossível atacá-la com sucesso. É, na verdade, afirmar que o custo de um ataque com sucesso é mais alto que o valor de um eventual ganho que o atacante venha a ter com um ataque bem sucedido.

Atualmente, não existe nenhum tipo de mecanismo de segurança relacionado ao núcleo do protocolo USB, os dispositivos são livres para adicionar essa segurança no topo do protocolo. Sendo a segurança um requisito necessário na substituição de uma comunicação através de fios por uma comunicação *over-the-air*[5].

## 4.2 Modelo de Criptografia do WUSB

O método de criptografia recomendado pela especificação do protocolo WUSB é o AES de 128 bits no modo de operação CCM. O WUSB também suporta criptografia de chave pública, entretanto, ela só é utilizada para a associação do dispositivo.

Os dispositivos que implementam criptografia de chaves públicas, têm que suportar o algoritmo RSA com chaves de 3071 bits e o algoritmo de *hash* SHA-256, mantendo assim um nível equivalente de segurança ao AES 128 CCM utilizado após a troca de chave inicial nas comunicações.

[5] O termo *over-the-air,* refere-se a comunicação sem-fio.

Várias chaves são utilizadas na comunicação segura do dispositivo com o *host* no padrão WUSB. As *Master Keys* têm vida longa e são geralmente utilizadas para compartilhamento do segredo de autenticação.

Para a associação de dispositivos através do método de chaves públicas, é utilizado um par de chaves (PK). Cada dispositivo, que usa chave pública, deve conter a sua própria chave. Caso a geração da chave ocorra na hora em que o dispositivo é ligado, esta deverá ser feita em menos de 3 segundos, tempo esperado para que o dispositivo esteja pronto para o funcionamento segundo as especificações do WUSB [30].

A chave de conexão (CK) é a chave secreta utilizada para estabelecer conexões. Ela é gerada pelo *host* e distribuída para os dispositivos juntamente com o CHID (*Connection Host Identification* - Identificador de conexão de *host*) e a CDID (*Connection Device Ideintification* - Identificador de conexão do dispositivo) no momento da associação. É importante observar que o *host* não deve mandar a mesma CK para vários dispositivos, a menos que, seja com o propósito de depuração. O envio da mesma CK para vários dispositivos aumenta a probabilidade de descoberta por um atacante.

As chaves de seção são utilizadas somente durante o tempo de conexão de um dispositivo, sendo descartadas logo em seguida.

As chaves do tipo GTK (*Group Temporal Key*) não são utilizadas para cifra de dados; são usadas apenas para checar a integridade das mensagens. Elas são de conhecimento de todos os dispositivos que estão conectados a um determinado *host.* O *host* utiliza essa chave quando faz alguma comunicação para todos os dispositivos que estão conectados a ele.

## 4.3 *Connection Context*

Para que haja qualquer comunicação segura, tanto o *host* quanto o dispositivo mantêm três informações em comum, CHID, CDID e CK. Essas informações devem ser de conhecimento dos dois, permitindo que não seja necessário fazer uma nova associação para se estabelecer uma nova conexão. Esses três elementos juntos são conhecidos por *Connection Context* (CC) ou contexto de conexão.

Alguns dispositivos têm suporte a aceitar mais de um CC, neste caso o dispositivo deve armazenar tantas quantas forem as solicitações que ele puder. Os que não possuem tal suporte devem sobrescrever a associação com a mais recente. Os CCs só devem ser trocados através de uma canal seguro, pois, a interceptação dessa informação compromete a segurança, uma vez que o detentor dela pode conectar e desconectar do *host* sem autorização prévia.

Um dispositivo deve ser capaz de estabelecer uma conexão com qualquer *host* que ele já tenha se associado. Através de mensagens, o *host* comunica a sua existência para possíveis dispositivos. Essas mensagens são enviadas ininterruptamente pelo *host.*

Até que a conexão seja considerada operacional, o *host* e o dispositivo devem completar a fase de autenticação e associação. Enquanto isso não ocorre, o dispositivo não é reconhecido pelo barramento USB.

### 4.3.1 Associação com Chave Simétrica

O processo de associação *Out-of-Band* pode ocorrer de diversas maneiras diferentes. O princípio básico consiste no envio do CC para um dispositivo através de um canal seguro fora do meio normal de comunicação WUSB. O processo pode ocorrer via USB com fio, a associação ocorre quando o dispositivo USB é ligado ao *host*. Nesta hora a confirmação do usuário é necessária

para continuar o processo, essa pode ser feita via software. Após o dispositivo ser desligado do *host* é possível a conexão via WUSB, onde através de solicitação do usuário, o dispositivo inicia o processo de associação. Depois de iniciado, o dispsitivo envia ao *host* uma requisição de conexão. Após a resposta da requisição, eles fazem a troca de chave inicial (*handshake)* e o dispositivo passa a estar operacional, como ilustrado na Figura 16.

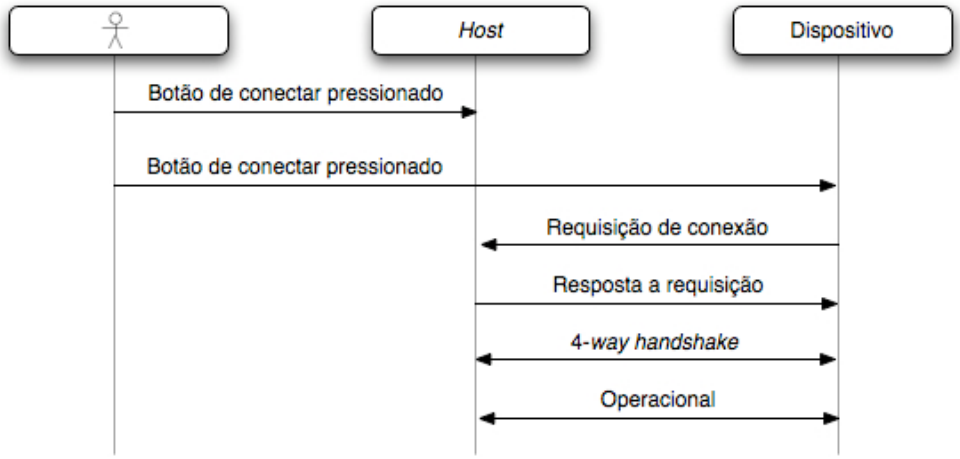



**Figura 16.** Processo de associação [30].

### 4.3.2 Associação com Chave Pública

O processo de associação através de chave pública começa quando o usuário, através do *host*, anuncia e aceita novas conexões. O dispositivo por sua vez, ao receber tal informação, responde com uma requisição de conexão. O *host* completa o requerimento e notifica a entidade de segurança que um novo dispositivo deve ser processado. Depois de ser enumerado pelo *host*, é enviado para o dispositivo uma requisição contendo a chave pública do *host* e solicitando a chave pública do dispositivo.

Nesse ponto, o usuário é envolvido e é dele a confirmação de que o *host* e o dispositivo receberam as chaves corretamente. Essa validação é necessária para provar que tanto o *host* quanto o dispositivo são quem dizem ser e que não se trata de um agente malicioso, ou seja, garantindo a autenticidade.

Depois da validação da chave do dispositivo, o *host* deve usar a chave pública do dispositivo para cifrar e distribuir o CC.

# Capítulo 5

# Análise da Segurança no WUSB

Este Capítulo trata da análise do modelo de segurança adotado pelo padrão WUSB. A Seção 5.1 descreve alguns aspectos teóricos do processo de criptografia. Na Seção 5.2, é possível encontrar uma análise dos métodos de associação de dispositivos com *hosts*. A Seção 5.3 contém uma análise da implementação de código aberto do protocolo WUSB, a qual se encontra disponível para a comunidade de usuários do Linux em [31].

## 5.1 Padrão de Criptografia do WUSB

O algoritmo de cifra AES possui inúmeras vantagens como: ser um algoritmo de criptografia que exige um baixo custo computacional, possui alta velocidade para realização de operações, entre outras. Essas são algumas das características que definem o AES como algoritmo padrão de criptografia do WUSB.

A configuração do AES utilizada no WUSB é feita com chaves de 128 bits no modo de operação CCM. Essa configuração torna o algoritmo capaz de garantir a confidencialidade e a autenticidade da informação trafegada no canal de dados [30], [32].

Como já dissemos, a confidencialidade na comunicação de dispositivos USB com fio é garantida pelo próprio cabo de conexão, uma vez que a propagação é difundida unicamente pelo fio, impedindo a captura externa. O uso de criptografia (no caso, o AES) é necessário nas conexões sem fio para dificultar a captura de informações transmitidas pelo dispositivo.

### 5.1.1 Análise do modelo de criptografia adotado na Comunicação WUSB

Nessa abordagem, destacaremos dois tipos de ataques: os ataques passivos e os ataques ativos. Nos ataques passivos, o invasor age sem ser percebido, se aproveitando dos dados que consegue captar. No ataque ativo, o invasor tem a capacidade de manipular as informações no canal, criando um ambiente propício para o seu ataque.

A comunicação WUSB é protegida a ataques do tipo texto claro escolhido [17]. Esses ataques presumem que o atacante, por possuir a capacidade de gerar dados cifrados a partir de texto claro escolhido, pode associar, através de heurística, as diferenças a uma chave. Como no caso da comunicação WUSB o atacante age passivamente, para ele não é possível a manipulação do texto a ser cifrado.

O AES com CCM não é vulnerável a ataques do tipo texto claro conhecidos [17]. Tais ataques, ao contrário do texto claro escolhido, podem ser realizados passivamente. Porém o modo CMM, por usar o modo *Counter*, evita que a chave privada seja capturada.

O modo *Counter*, além de usar a chave para fazer a cifra, usa também o *nonce* que é um número que representa unicamente um bloco. Um fator que depende da quantidade de blocos que já foram cifrados que, geralmente, não é de conhecimento do invasor.

No caso do WUSB, o *nonce* utilizado é uma composição de valores, que consideram os dados: SFN, TKID, *DestAddr* e *SrcAddr* de acordo com a Tabela 3.

**Tabela 3.** Dados que são considerados na geração do *nonce* utilizado pelo WUSB.

| *Offset* | Campo | Tamanho | Descrição |
|---|---|---|---|
| 0 | *SFN* | 6 | O número do *frame* seguro associado a essa mensagem |
| 6 | *TKID* | 3 | Referência da chave utilizada para cifrar/decifrar essa mensagem |
| 9 | *DestAddr* | 2 | Endereço do dispositivo de destino |
| 11 | *SrcAddr* | 2 | Endereço do dispositivo de origem |

A utilização do *nonce*, também garante que o resultado da cifra de dois dados iguais produzam resultados diferentes, adicionando um maior caráter de aleatoriedade ao processo e, dessa forma, introduzindo um complicador a mais no código, dificultando a criptoanálise.

A resistência a ataques ativos e não somente a passivos, como demonstrado anteriormente, também é uma característica desse modelo adotado pelo padrão WUSB. Protegendo a comunicação de ataques como *Men-in-the-middle,* MITM [17], que caracteriza-se por explorar os primeiros passos da comunicação cifrada, antes que um elo de segurança seja estabelecido. Nesses casos especiais, o invasor, ou homem do meio, se apresenta para *A* como *B* e como *B* para *A*, ficando exatamente no meio da comunicação, a par de todas as informações que passam pelo o canal. A Figura 17 mostra com mais detalhes um ataque de MITM.

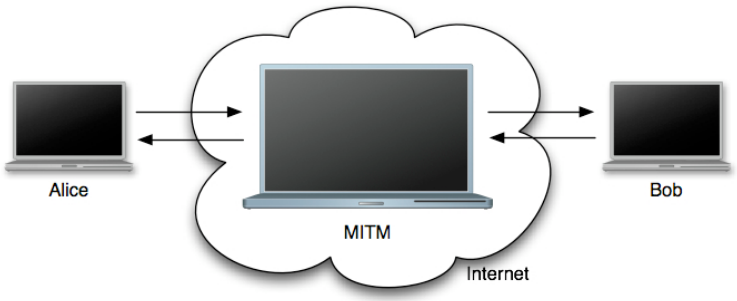



**Figura 17.** Exemplo de ataque de MITM: Bob tenta mandar uma mensagem para Alice; entretanto, o invasor coloca-se no meio da conexão, fingindo ser Alice para Bob e Bob para Alice.

### 5.1.2 Garantia de Autenticidade

O modo de operação CBC-MAC, que é utilizado pelo CCM, garante a autenticidade das informações no canal de comunicação. O processo de decifragem/validação do MAC determina se, de fato, o bloco veio do processo de cifragem, onde foram utilizados: uma chave *K*, um bloco de dados *D* e um *nonce*. Isso garante se o bloco recebido veio do transmissor legítimo ou não.

Para qualquer texto cifrado de tamanho *M* pelo menos *M* bytes correspondem a um MAC cifrado, o resto da mensagem pode ser considerado como *payload* (no caso, os dados cifrados). No WUSB, essas informações estão dispostas de acordo com a Tabela 4.

**Tabela 4.** Distribuição do cabeçalho MAC mais *payload* num bloco de dados WUSB.

| *Offset* | Campo | Tamanho | Tipo | Descrição |
|---|---|---|---|---|
| 0 | cabeçalho MAC | 10 | Record | Campo de 10 bytes obrigatoriamente preenchido com o cabeçalho do MAC. |
| 11 | *payload* | N | Bitmap | Refere-se aos dados cifrados. |

O processo de verificação checa a corretude do MAC em relação ao *payload* e às informações adicionais: *nonce* e bloco de dados. No ato da validação, duas são as respostas esperadas: "inválido" ou o próprio MAC.

- Quando inválido, tanto o *payload* quanto os dados acessórios não podem ser validados de forma independente; apenas é possível aferir que eles não foram gerados a partir de um emissor confiável.
- Quanto o resultado é o próprio MAC, de acordo com o *design* do modelo, é possível afirmar que os dados são autênticos. Entretanto, tal afirmação não pode ser considerada absolutamente verdadeira, afinal, um atacante pode gerar uma mensagem que passe por essa validação.

A probabilidade de um atacante ter sucesso na validação de uma mensagem é inversamente proporcional ao tamanho do MAC. É claro que um atacante pode enviar dados aleatórios a fim de obter, ao menos, uma mensagem aceita pela validação.

### 5.1.3 Possibilidade de Conexões Acidentais

Após a associação, fica a critério do usuário autorizar ou não a conexão, sendo essa autorização concedida ao dispositivo através do seu nome. Em um cenário onde existem *n* dispositivos associados, o usuário pode eventualmente selecionar o errado, ocasionando negação do serviço, caso alguém já esteja utilizando o dispositivo. Os dispositivos conectados ao DWA agora estariam disponíveis para o novo *host* permitindo que o usuário do novo *host* acesse dados pessoais do usuário do primeiro *host* como, por exemplo, um *pen-drive*.

É imprescindível que o usuário só associe o seu dispositivo WUSB em *hosts* que ele realmente confie, até porque não existe na definição do protocolo de associação, uma interface para deleção da associação. A delação só e possível de ser feita no *host* que, se maliciosamente configurado, não deletará de fato as informações do CC, única informação necessária para o *host* re-estabelecer uma comunicação com o dispositivo.

Um outro cenário possível de acontecer é a conexão a um DWA maliciosamente posto a fim de confundir o usuário, tendo conectado a ele dispositivos de classe *mass storage* (*pen-drives*) que podem injetar vírus, *malwares* ou ainda *spywares,* sem que o usuário do *host* perceba.

## 5.2 Análise dos métodos de associação

Conforme detalhado anteriormente, o método de associação entre o dispositivo e o *host* é bastante importante para que uma comunicação seja estabelecida de forma segura, além de garantir que o processo está realmente seguindo o que está descrito na especificação, sendo isso vital para a segurança do processo.

Como garantia de corretude da associação, esta Seção visa a apresentar os resultados da análise entre a associação de um dispositivo WUSB (WUSB Hub do fabricante IOGear) e um *host* que utiliza um HWA também da IOGear. O *software* de associação analisado foi fornecido pelo próprio fabricante e pode ser encontrado em [33].

O processo a ser analisado é uma visão das informações trocadas entre o dispositivo e o *host* através da USB. Esses dados foram obtidos através do uso da ferramenta de captura UsbSnoop que pode ser encontrada em [34].

### 5.2.1 Associação

O processo de associação tem o objetivo de garantir a troca da chave de conexão (CK). Esse processo ocorre da seguinte forma: o dispositivo é conectado através da USB de um *host* e esse dispositivo, após ser identificado, passa por um processo de reconhecimento, onde aguarda a autorização do usuário que dará início ao processo de associação.

Como parte da análise da segurança do WUSB, o processo de associação foi detalhado em 4 blocos referentes a cada requisição USB. O primeiro bloco refere-se à requisição de informação de associação (GET_ASSOCIATION_INFORMATION) e está ilustrado na Figura 18.

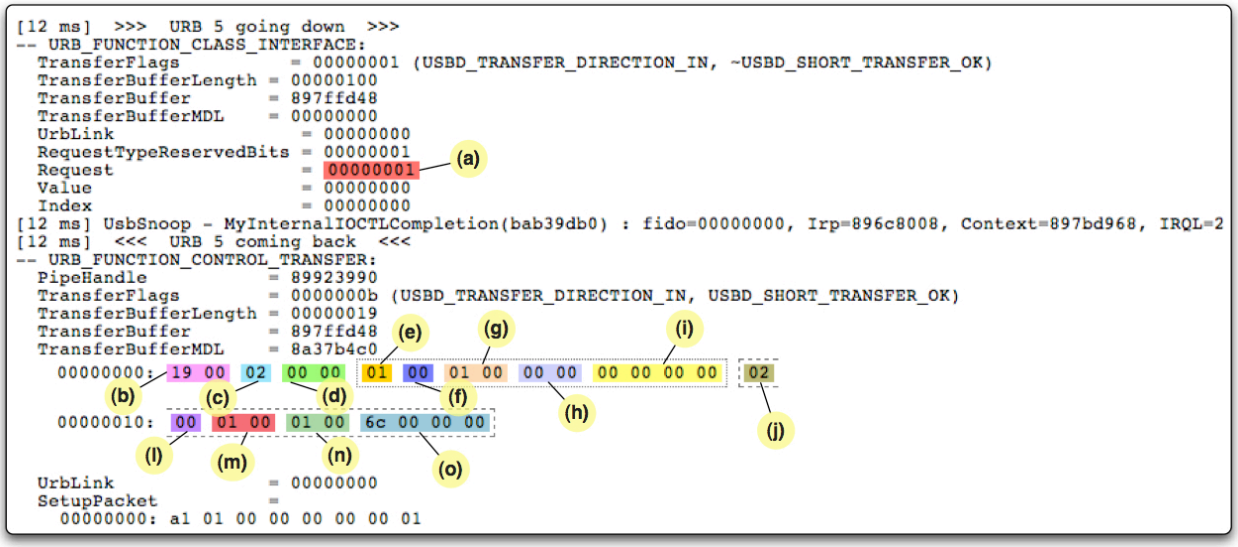



**Figura 18.** Primeira troca de mensagem entre o *host* e o dispositivo. (a) Tipo da comunicação: GET_ASSOCIATION_INFORMATION. (b) Tamanho da estrutura. (c) Número de requisições. (d) Valor pré-definido. (e) Valor do índice. (f) Byte reservado. (g) Valor pré-definido. (h) Tipo da requisição. (i) Tamanho da informação associada a requisição. (j) Valor do índice. (l) Byte reservado. (m) Valor pré-definido. (n) Tipo da requisição. (o) Tamanho da informação associada a requisição.

Esse segundo bloco de informação contém os dados, a respeito do *host*, que são enviados para o dispositivo. Esse bloco está ilustrado na Figura 19.

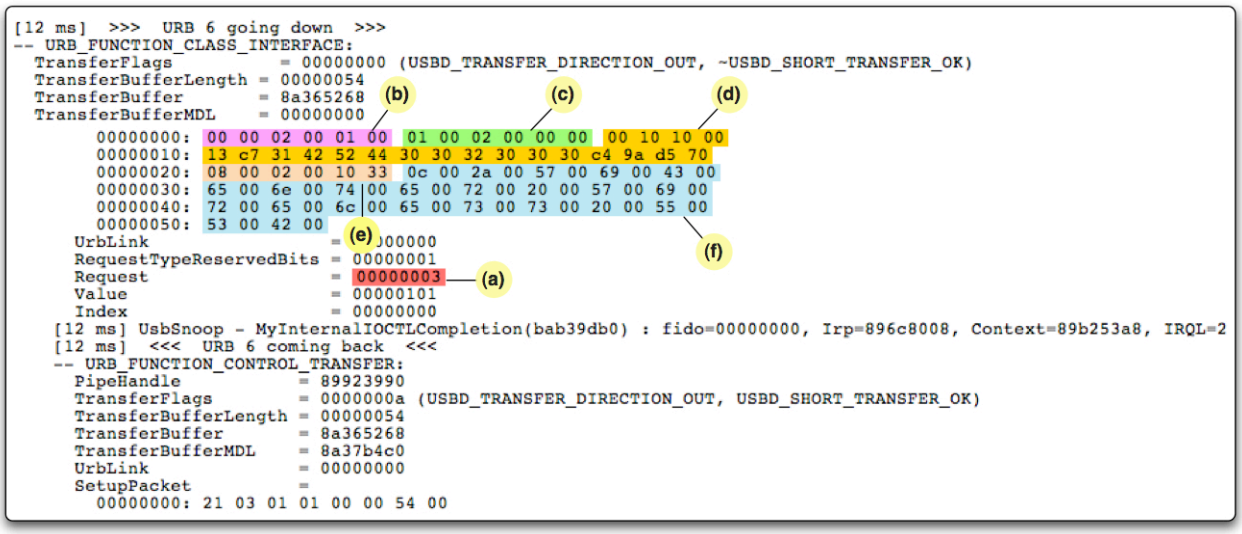



**Figura 19.** Segunda troca de mensagem entre o *host* e o dispositivo. (a) Tipo da comunicação: HOST_INFO. (b) Tipo da associação. (c) Sub-tipo da associação. (d) CHID. (e) ID da linguagem utilizada. (f) Nome amigável do *host*.

O terceiro bloco do processo de associação contém informações a respeito do dispositivo e está ilustrado na Figura 20.

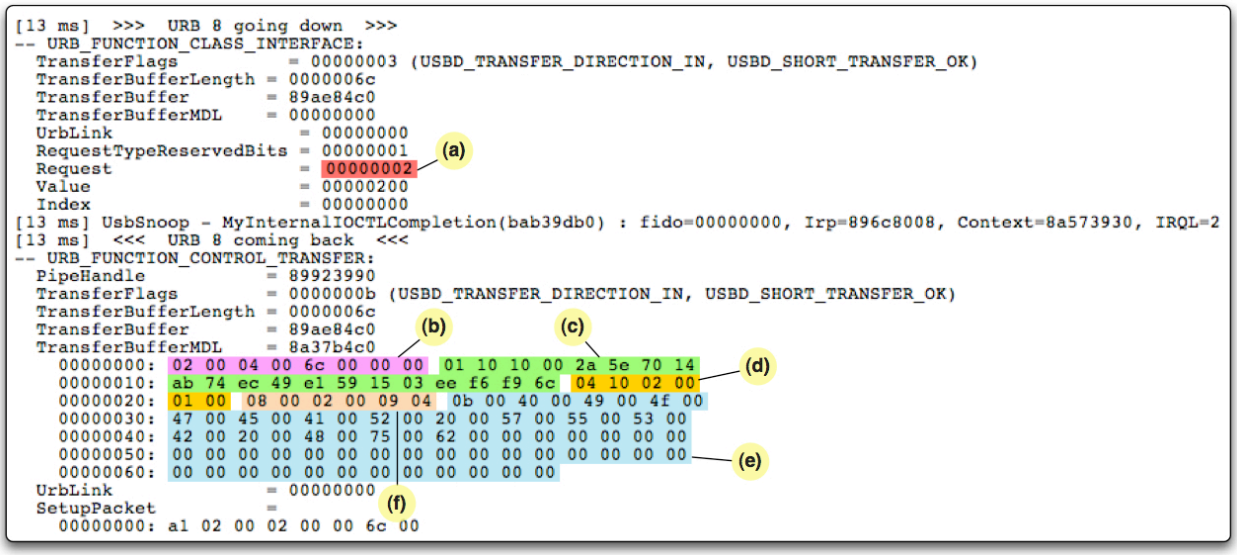



**Figura 20.** Terceira troca de mensagem entre o *host* e o dispositivo. (a) Tipo da comunicação: DEVICE_INFO. (b) Tamanho desta estrutura. (c) CDID. (d) BandGroups. (e) Nome amigável do dispositivo. (f) ID da linguagem utilizada.

O quarto bloco contém a quarta e última troca de mensagem entre o *host* e o dispositivo. É nesse bloco que ocorre a confirmação da associação através da passagem da CK. O bloco está ilustrado na Figura 21.

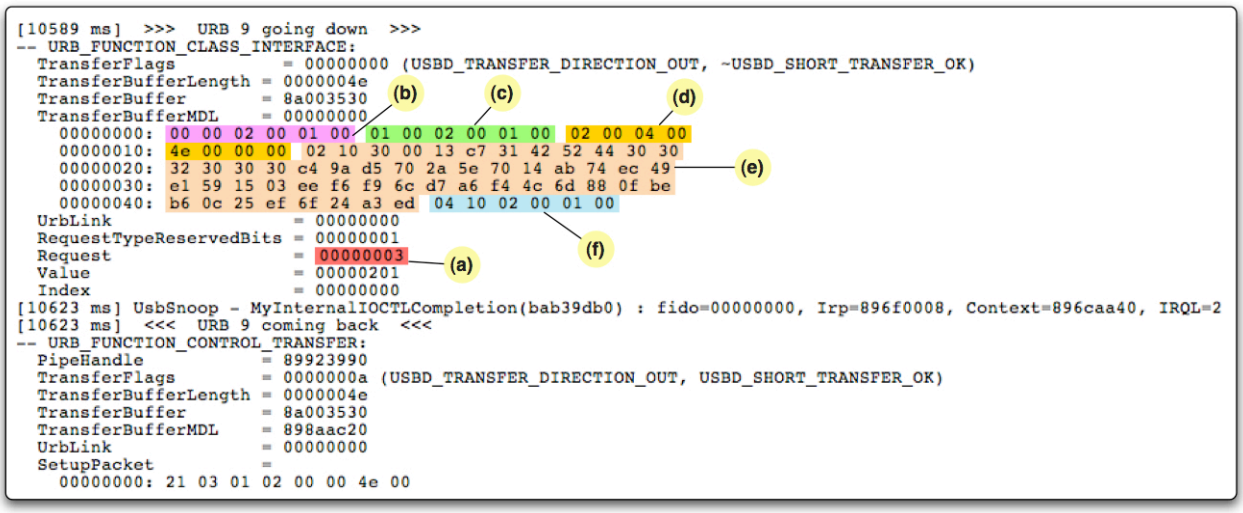



**Figura 21.** Quarta e última troca de mensagem entre o *host* e o dispositivo. (a) Tipo de comunicação: SET_ASSOCIATION_REQUEST. (b) Tipo da associação. (c) Sub-Tipo da associação. (d) Tamanho da estrutura de dados. (e) CC, *Connection context.* (f) BandGroups.

Para um melhor entendimento, é valido ressaltar que todos os dados numéricos representado nessa análise estão no formato *little-endian*[6]. As *strings* estão representadas no formato *Unicode*[7], por exemplo, a Figura 20, legenda *e*, representa: "IOGEAR WUSB Hub".

Com essa análise, é possível constatar que os equipamentos e softwares testados estão dentro da especificação do padrão WUSB e, por isso, podem ser considerados seguros. Além disso, pode-se mostrar em detalhes como ocorre o processo de associação, fato esse, que já representa uma grande contribuição.

Não fica claro, entretanto, como o dispositivo armazena as chaves, uma vez que ele tem suporte a armazenar mais de uma e não existe uma interface para listá-las – requisito da especificação.

## 5.3 Análise do Suporte ao WUSB no Linux

Apesar do WUSB não ser uma tecnologia tão recente (especificada desde maio de 2005) o seu uso não é tão popular. A tecnologia ainda não encontra suporte nativo nos sistemas operacionais. Os *softwares* de suporte, fornecidos pelos fabricantes dos *hardwares*, ainda encontram-se bastante instáveis, além de não suportarem todos os tipos de transferências USB, como por exemplo: as transferências de *stream*.

Essa análise visa a identificar e corrigir disparidades entre o modelo proposto pela especificação e a implementação. Para tal validação, primeiro foi necessário garantir o

[6] O termo *little-endian* refere-se a ordenação de bytes usado para representar algum tipo de dado na memória. Dizer que o dado está armazenado num formato *little-endian* e afirmar que se byte menos significativo esta no endereço mais baixo. Os outros bytes seguem em ordem de significância crescente [35].

[7] *Unicode* é um padrão que permite aos computadores representar e manipular, de forma consistente, texto de qualquer sistema de escrita [36].

funcionamento dos HWA. Nos dois dispositivos testados foi necessário o uso de um arquivo de *firmware*[8]. Sendo o arquivo de *firmware* do distpositivo da Intel liberado para *download* em [38].

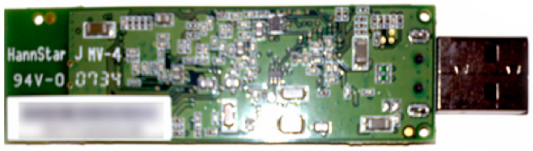
Após a análise do suporte ao HWA no Linux, foi analisado o suporte à associação dos dispositivos com o *host* através de cabo. Essa última análise também visa a verificar a corretude da implementação do modelo proposto em relação às especificações do WUSB e do WUSB *Association Model* [39].

### 5.3.1 Análise do Suporte ao HWA

O *firmware* do HWA da IOGear, até então não testado no Linux, não é liberado pelo fabricante. Entretanto, o mesmo pode ser encontrado dentro dos arquivos distribuídos junto aos *drivers*. Utilizando novamente a ferramenta de captura dos dados da USB, UsbSnoop [34] foi possível achar o bytes iniciais e finais do *firmware*, que, por conseqüência, permitiu achar em que arquivo o mesmo estava inserido.

Apesar dos dispositivos possuírem fabricantes diferentes, fisicamente eles são bastante semelhantes, como pode ser visto na Figura 22. Todos os dois dispositivos utilizam um *chipset* fabricado por uma empresa chamada Alereon, a referência do *chipset* é AL4200.

(a)




(b)

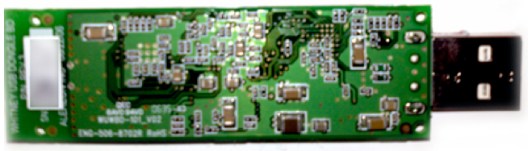


**Figura 22.** Dispositivos HWAs. (a) Dispositivo do fabricante IOGear. (b) Dispositivo do fabricante Intel.

Dadas as semelhanças, testes foram feitos para averiguar se os dispositivos são capazes de trabalhar com *firmwares* de outros fabricantes. Os resultados podem ser encontrados na Tabela 5.

**Tabela 5.** *Firmawares* suportado pelos dispositivos HWA.

| Dispositivo \ *Firmware* | **i1480-usb-RC1.3** | **i1480-usb-RC1.2** | **IOGear** |
|---|---|---|---|
| **HWA Intel** | Suportado | Suportado | Parcialmente suportado |
| **HWA IOGear** | Suportado | Suportado | Parcialmente suportado |

Utilizando o *firmware* da IOgear, os dispositivos funcionam. Entretanto, o *firmware* não provê suporte ao WLP, *Wimedia Link Protocol*. O WLP permite a comunicação IP (*Internet Protocol*) através dos HWA. WLP não faz parte do escopo deste trabalho e as suas especificação ainda não foram concluídas.

[8] *Firmware* é o termo usado para se referenciar um software que roda embarcado em um *hardware* [37].

Após a conclusão de quê *firmwares* utilizar, foi necessário modificar o código do suporte a upload do *firmware,* pois o mesmo apresentava diversos *warnings* e ignorava alguns erros que deveriam ser tratados. Esses foram corrigidos, a fim do código estar alinhado com o estilo de programação do *kernel* [40]. É importante seguir esse estilo, para que através da submissão do *patch*[9] o código possa ser analisado por outros desenvolvedores. Caso não esteja dentro dos padrões, o código sequer é analisado.

Esses são os *patches* inicialmente submetidos para a comunidade *linux-uwb*:

- [linux-uwb] [PATCH] dfu: *Use %zu instead of %d when dealing with size_t*
- dfu: *Makes i1480_cmd_reset(), i1480_mpi_write() and i1480_mpi_read() static instead of global*

Após as correções, o usuário deverá obter no seu *log* uma saída como a ilustrada na Figura 23. Essa saída deve ser vista logo após o dispositivo ser ligado ao sistema.

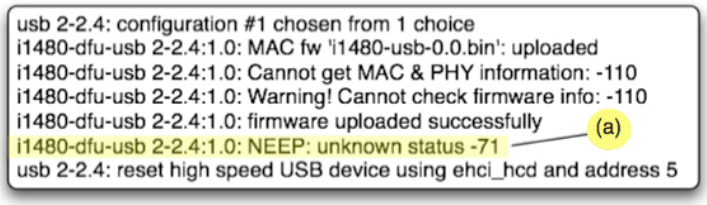



**Figura 23.** Trecho do *log* de sistema, após d dispositivo ser ligado ao sistema, onde (a) representa uma mensagem de erro após o envio do *firmware.*

Na Figura 23, na legenda *a*, pode-se observar que ainda persiste uma mensagem de erro; esse erro não foi tratado, pois, mesmo após discussão na comunidade, não se chegou a nenhuma conclusão da exata origem do mesmo. O erro poderia apenas representar um problema no *hardware,* que não atrapalha o funcionamento final do dispositivo.

Com o fim da análise do suporte ao HWA no Linux e com *patches* aceitos pela comunidade, o dispositivo está apto a funcionar de acordo com as especificações.

Sabendo-se que o dispositivo DWA, que foi utilizado neste trabalho, necessita realizar sua associação através de cabo, este é, naturalmente, o próximo passo a ser analisado.

### 5.3.2 Análise do Suporte à Associação

Na comunidade do Linux UWB já havia suporte para CBA, o CBAF (*Cable Association Framework*). Esse modelo apenas apresentava uma interface de comunicação entre a *user space*[10] e o dispositivo. Essa interface é realizada através da interface de comunicação *sysfs* que provê uma hierarquia de diretórios e arquivos especiais que podem ser listados pelos usuários de uma máquina, permitindo a leitura e escrita desses arquivos especiais. A leitura e escrita nesses arquivos virtuais, na verdade, são gatilhos para funções em *kernel space*[11], na escrita os dados são passados para *kernel space* e referenciados através de um ponteiro.

---

[9] *Patch* é um arquivo texto que contém as diferenças da alteração de um arquivo, estas diferenças podem ser geradas com o uso do comando diff [41] do Unix e aplicadas com um programa do mesmo nome, *patch* [42].

[10,11] O Linux cria uma barreira virtual entre a memória que esta em *Kernel Space* e a memória de *User Space*, em *Kernel space* esta tudo de *Kernel, Kernel extension* e alguns *drivers* de dispositivos, o resto roda em *User space* [43].

O estilo de código do Kernel recomenda que não devem existir entradas para dois tópicos diferentes em apenas um arquivo *sysfs*, sendo isso um problema para implementação do CBAF que em sua implementação original recebia múltiplos parâmetros como forma de entrada. Por exemplo, o parâmetro *wusb_cc* recebia como entrada CK, CHID e BandGroups.

Novamente, para atender aos padrões de codificação do Kernel, foram implementados *patches* para correção desses arquivos *sysfs*. Nesse conjunto de *patches*, também foram corrigidos problemas na entrada do BandGroups, que agora refere-se ao BandGroups do dispositivo e não do *host* como acontecia antes.

Após o CBAF estar organizado da forma correta, ficou claro constatar que havia um problema na implementação da função que recebe informações do dispositivo (Figura 20). Ao receber os dados com informações sobre o dispositivo, através de um conjunto de bytes, a implementação os reparte de maneira correta. Entretanto, utiliza a função *strncpy* [44] que, ao receber um conjunto de bytes, interpreta o "00" como final de um conjunto de caracteres. Como os dados repassados pelo dispositivo são seguidos sempre por um byte "00", o nome do dispositivo fica resumido ao seu primeiro caractere. Essa é uma falha grave, uma vez que o usuário pode ter mais de um dispositivo fazendo *Cable Association* e não conseguiria, assim, aferir quem é quem, se as primeiras letras forem iguais.

Também foram adicionados meios de depuração. Os meios anteriores também não seguiam os padrões aconselhados pela comunidade do *Kernel*. Os novos meios de depuração estão compatíveis com o modelo proposto e também ganharam entrada no menu de opções de compilação do Kernel para serem ativados ou não na compilação do módulo ou do Kernel.

### 5.3.3 Testes

Para testar os *patches* produzidos, foram utilizados 2 computadores Dell, Xeon 3.2 GHz com 2 Gigas de memória RAM e dois *notebooks* IBM ThinkPad T41 com processador Pentium M de 1.6 e 1 Giga de memória RAM.

Cada uma das máquinas de teste possui *Kernel* nas versões:

- 2.6.24
- 2.6.25
- 2.6.25-rc6
- 2.6.26-rc1
- 2.6.26-rc2

Nas máquinas Dell, foram instalados os *kernels* citados com suporte à instruções de 64 bits.

Como é comum no processo de testes/depuração, as máquinas podem "travar" isto acontece porque, em geral, o espaço de memória do Kernel não tem proteção; qualquer corrupção na memória acarreta em um congelamento que muitas vezes impede o testador de identificar a possível causa e de ler mensagens de depuração. Para que isso não ocorra, é necessária a utilização de um cabo de comunicação serial para que um console do computador possa ser exportado para outro computador, onde as mensagens do sistema são transferidas diretamente. No caso de um erro, a mensagem de depuração é enviada remotamente antes que a máquina pare de funcionar.

A etapa de testes é de extrema importância para garantir o bom funcionamento do WUSB uma vez que na ocasião de um erro no Kernel o usuário final não seria capaz de identificá-lo. Todos os *patches* produzidos no decorrer deste trabalho foram devidamente testados e homologados.

### 5.3.4 Homologação

A homologação ocorre através de uma hierarquia de 3 níveis, sendo o primeiro deles no mesmo computador onde são realizados os testes. Uma versão limpa do sistema operacional – que não foi utilizado para desenvolvimento – é utilizada para realização de uma bateria de testes. Uma vez que os *patches* passam por essa bateria são enviados para outra equipe de desenvolvimento que tem por função analisar o código, questões de estilo, identação e padronização, além de avaliar se os mesmos estão funcionando conforme deveriam.

Essa mesma equipe é responsável pela instalação do seu próprio ambiente, evitando problemas de vícios de instalações e aumentado a heterogeneidade do ambiente de testes.

A terceira e última fase de homologação acontece pela comunidade do projeto, onde o *patch* é submetido à análise de todos os envolvidos (usuários, desenvolvedores, pesquisadores). Depois de recebido os devidos comentários e realizadas as argumentações, o *patch* é finalmente aprovado pelo mantenedor da comunidade, que adiciona o mesmo na árvore de desenvolvimento oficial do projeto. Neste momento, o código torna-se oficialmente reconhecido pela comunidade, e fará parte de um *release*[12], de acordo com o cronograma do projeto.

[12] Entende-se por *release* uma versão de *software* que será disponibilisada para o usuário final.

# Capítulo 6

# Conclusões e Trabalhos Futuros

É notório que os usuários necessitam cada vez mais de mobilidade e, aliada a essa mobilidade, encontram-se a rapidez de acesso e agilidade nas comunicações. A USB sem-fio ou WUSB foi idealizada para suprir essa necessidade. Apesar do grupo de promotores da WUSB ter se formado em 2004 e de sua especificação ter sido finalizada em 2005, foi recente o avanço da tecnologia que tornou viável o uso comercial da WUSB. Também com esse avanço outras necessidades surgiram, como: implementação de *drivers* de dispositivos para sistemas operacionais que não eram suportados inicialmente, validação da segurança das aplicações fornecidas pelos fabricantes de dispositivos etc.

Nesse contexto, este trabalho sugere uma análise mais apurada da segurança da informação nos dispositivos WUSB e de seu protocolo de comunicação. Para tal, foi realizado primeiramente um estudo acerca da tecnologia envolvida, criando uma base de conhecimento necessária para o início da análise. Para este estudo foram utilizados materiais referentes à criptografia e à tecnologia USB, além das especificações sobre WUSB.

Após serem devidamente instalados, os *drivers* dos dispositivos foram analisados, levando em consideração: a fidelidade aos padrões de codificação do Kernel e ao modelo de especificação WUSB. Para a análise, também foi necessária a extração do arquivo de *firmeware* do dispositivo IOGear. Foi possível concluir que o código dos *drivers* necessitava de algumas mudanças para atender os aspectos analisados. Duas mudanças de código foram implementadas, ocasionando a geração de dois *patches*, com o intuito de atender aos padrões de codificação do Kernel e o modelo de especificação WUSB.

Uma análise posterior trata de aferir se a implementação do *Cable Association* ou CBA no sistema operacional Windows, fornecida pelo fabricante IOGear, está fidedigna às especificações do WUSB e do WUSB *Association Model*. A partir dessa análise, foi possível concluir que a implementação de fato atende a todas as especificações.

Também for realizada uma análise do *Cable Association* no Linux, onde foi constatada a necessidade de pequenas mudanças no código do CBA. Essas mudanças geraram uma série de *patches* de correção.

A proposta do modelo de segurança WUSB é fazer a equivalência da segurança dos dispositivos com-fio com dos dispositivos sem-fio, e demonstra-se bastante eficaz nesse ponto. Entretanto, alguns pontos são falhos, como a possibilidade de conexão com o dispositivo errado e a falta da opção de deleção de uma associação. Na Tabela 6 encontra-se um comparativo entre a USB com fio e a USB sem fio em relação a segurança da informação.scp

**Tabela 6.** Tabela comparativa entre a segurança da USB com fio e sem fio

| | **USB com fio** | **USB sem fio** |
|---|---|---|
| Ataques passivos | Impossível | Fácil |
| Integridade | Garantida | Garantida |
| Autenticidade | Garantida | Garantida |
| Conexão acidental | Impossível | Fácil |
| MITM | Impossível | Desconhecido |

O fato do AES 128bits com CCM ser o único algoritmo citado pelas especificações é um problema, uma vez que o poder computacional cresce cada vez mais, auxiliando a criptoanálise. Apesar de hoje não existirem técnicas conhecidas para a quebra da cifra AES nesse modo de operação, poderá existir em um futuro próximo e isso acarretará num legado de dispositivos inseguros. Se não for estabelecido um meio dos dispositivos se inter-operarem, suportando diversos algoritmos de cifras, não será possível ter num mesmo *host* dispositivos, utilizando o AES 128 bits e outro algoritmo.

Um ponto muito forte no modelo adotado é o fato das conexões já começarem protegidas. O próprio aviso do *host* para comunicar que está ativo (MMC) já se encontra protegido com a GTK. A GTK é utilizada somente para validar o MMC, a informação do GTK é transmitida para o dispositivo depois que o mesmo se conecta com o *host* pela primeira vez; a partir de então, ele passa a validar as informações no MMC.

De forma geral, foi percebido ao longo deste trabalho que a qualidade do código do WUSB no Linux deixa a desejar quando contrastado com os padrões especificados pela comunidade do *Kernel* sendo esse um ponto no qual tivemos que atacar, para contarmos com a homologação da comunidade.

## 6.1 Dificuldades Encontradas

Durante este trabalho, algumas dificuldades foram encontradas, como:

- A dificuldade na aquisição dos dispositivos WUSB, devido à pouca quantidade de fabricantes, à venda limitada aos países que possuem a banda utilizada pelo UWB não regulamentada e por diversos embaraços alfandegários.
- Falta de documentação e de cooperação dos fabricantes de alguns dispositivos.
- A falta de padronização no código do WUSB no Linux, tornando difícil o entendimento do mesmo.

## 6.2 Trabalhos Futuros

É clara a necessidade de análise de outras partes do código do WUSB no Linux, bem como, a adição de suporte a outros dispositivos de diferentes fabricantes. Então, é possível prever a continuidade deste trabalho.

# Bibliografia

[1] BUCCI, Andrea, COSTA, Newton. A Interface USB: Uma Visão Geral. Universidade Estadual de Campinas, 2006. Disponível em: <www.ic.unicamp.br/~rodolfo/Cursos/mc722/2s2006/Trabalho/g17-apresentacao.pdf>. Último acesso em: 17 de março de 2008.

[2] Agere Systems. Disponível em: <http://www.agere.com>. Último acesso em: 23 de março de 2008.

[3] Hewlett-Packard. Disponível em: <http://www.hp.com>. Último acesso em: 23 de março de 2008.

[4] Intel. Disponível em: <http://www.intel.com>. Último acesso em: 23 de março de 2008.

[5] Microsoft Corporation. Disponível em: <http://www.microsoft.com/en/us/default.aspx>. Último acesso em: 23 de março de 2008.

[6] NEC. Disponível em: <http://www.nec.com>. Último acesso em: 23 de março de 2008.

[7] Philips Semiconductors. Disponível em: <http://www.nxp.com/>. Último acesso em: 23 de março de 2008.

[8] Sansung Eletronics. Disponível em: <http://www.samsung.com>. Último acesso em: 23 de março de 2008.

[9] KOLIC, Rafael. Wireless USB Brings Greater Convenience and Mobility to Devices. Intel's Corporate Tecnology Group, 2004. Disponível em: <http://www.deviceforge.com/articles/AT9015145687.html>. Último acesso em: 17 de março de 2008.

[10] ROCHA, Érico, BRUNO, Gaspare. Estudo e análise de vulnerabilidades de segurança na tecnologia Bluetooth. Centro Universitário Lasalle, Canoas, RS.

[11] KARYGIANNIS, Tom, OWENS, Les. Wireless Network Security 802.11, Bluetooth and Handheld Devices. NIST Special Publication 800-48, novembro de 2002.

[12] Federal Standard 1037c. ANS T1.523-2001, Telecom Glossary 2000. Disponível em: <http://www.atis.org/tg2k/>. Último acesso em: 16 de marco de 2008.

[13] BRAINARD, J., JUELS, A., RIVIEST, R., SZYDLO, M., YUNG, M. Fourth-Factor Authentication: Somebody You Know. ACM CCS, pp. 168-78, 2006.

[14] SOUSA, Rafael, PUTTINI, Ricardo. SSLeay/OpenSSL Tutorial. Disponível em: <http://www.redes.unb.br/security/ssleay/tutorial.html>. Último acesso em: 17 de março de 2008.

[15] ANDERSSON, Fredik. ISO/IEC 17799 Compliant? Master Thesis, Department of Computer and Systems Sciences, Stockholm's University, dezembro de 2004.

[16] HARRIS, S. *All in One CISSP: Exam Guide.*California: McGraw-HILL/Osborne, 2005.

[17] MAO, W. *Modern Cryptography: Theory & Practice*. Nova Jersey: Prentice Hall PTR, 2004.

[18] BURNETT, S., PAINE, S. *Criptografia e Segurança: O Guia Oficial RSA*. Rio de Janeiro: Campus, 2002.

[19] GOMES, A. Criptografia Usando Protocolos Quânticos. Universidade Federal de Lavras, 2004. Disponível em: <http://bazar.ginux.ufla.br/index.php/MonosARL/article/viewPDF Interstitial/93/25>. Último acesso em: 14 de maio de 2008.
[20] NIST. Announcing the Advanced Encryption Standard (AES). *Federal Information Processing Standards Publications*, 2001.
[21] NIST. Data Encryption Standard (DES). *Federal Information Processing Standards Publications,* 1999.
[22] IBM, Disponível em: <http://www.ibm.com>. Último acesso em: 25 de maio de 2008.
[23] Rijnadael S-BOX, Disponível em: <http://en.wikipedia.org/wiki/Rijndael_S-box>. Último acesso em: 25 de maio de 2008.
[24] Nibble, Disponível em: <http://en.wikipedia.org/wiki/Nibble>. Último acesso em: 25 de maio de 2008.
[25] RIVEST, R. Lecture on MACs, 1997, Disponível em: < http://web.mit.edu/6.857/OldStuff/Fall97/lectures/lecture3.pdf >. Último acesso em: 25 de maio de 2008.
[26] USB Specification, Disponível em: <http://www.usb.org/developers/docs/usb_20_ 040908.zip>. Último acesso em: 25 de maio de 2008.
[27] USB Implementers Forum. Disponível em: <http://www.usb.org/developers>. Último acesso em: 25 de maio de 2008.
[28] Rede estrela. Disponível em: <http://en.wikipedia.org/wiki/Star_topology>. Último acesso em: 25 de maio de 2008.
[29] Linux Kernel. Disponível em: <http://www.kernel.org>. Último acesso em: 25 de maio de 2008.
[30] WUSB Specification. Disponível em: <http://www.usb.org/developers/wusb/ wusb_2007_0214.zip>. Último acesso em: 27 de maio de 2008.
[31] Linux UWB. Disponível em: <http://www.linuxuwb.org/>. Último acesso em: 25 de maio de 2008.
[32] NIST. Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality.
[33] IOGear Support. Disponível em: <http://www.iogear.com/support/dm/>. Último acesso em: 25 de maio de 2008.
[34] UsbSnoop. Disponível em: <http://usbsnoop.sourceforge.net/>. Último acesso em: 25 de maio de 2008.
[35] *Little-endian.* Disponível em: <http://en.wikipedia.org/wiki/Endianness>. Último acesso em: 25 de maio de 2008.
[36] *Unicode*. Disponível em: <http://pt.wikipedia.org/wiki/Unicode>. Último acesso em: 25 de maio de 2008.
[37] *Firmwares*. Disponível em: <http://en.wikipedia.org/wiki/Firmware>. Último acesso em: 25 de maio de 2008.
[38] Intel i1480 *Firmwares*. Disponível em: < http://www.linuxuwb.org/thewiki/Firmware_ for_the_Intel(r)_Wireless_UWB_Link_1480>. Último acesso em: 25 de maio de 2008.
[39] Association Models Supplement to the Certified Wireless Universal Serial Bus Specification. Disponível em: <http://www.usb.org/developers/wusb/ wusb_2007_0214.zip>. Último acesso em: 25 de maio de 2008.
[40] Kernel coding style. Disponível em: <http://lxr.linux.no/linux/Documentation/ CodingStyle>. Último acesso em: 25 de maio de 2008.
[41] Diff man page. Disponível em: <http://unixhelp.ed.ac.uk/CGI/man-cgi?diff>. Último acesso em: 25 de maio de 2008.

[42] Patch man page. Disponível em: <http://www.rt.com/man/patch.1.html>. Último acesso em: 25 de maio de 2008.

[43] CORBET, J., RUBINI, A., KROAH-HARTMAN, G. Linux Device Drivers, Estados Unidos: O'Reilly, 2005.

[44] strncpy man page. Disponível em: <http://linux.com.hk/penguin/man/3/strcpy.html>. Última acesso em: 25 de maio de 2008.