České Vysoké Učení v Praze – Fakulta Informačních Technologií

Smartcard Power Trace Measurement

Dokumentace

© Petr Vyleta [vyletpet@fit.cvut.cz]
1/11/2013

Smartcard Power Trace Measurement

Program je určený k měření odběru Smart karty při šifrování, k následnému prolomení použitého klíče korelační analýzou. Jeho součástí je tedy třída **Instrument**, který zajišťuje komunikaci s osciloskopem, třída **Scard** která zajišťuje komunikaci s kartou, a třída **Measurement**, která vše zastřešuje a obsahuje samotné měření, generování vstupních dat a uložení průběhů do souboru.

V následujícím textu popíšeme jednotlivé metody tříd, aby bylo možné program přizpůsobit, případně třídy použít k úplně jinému účelu v jiném programu.

Program vzhledem k použitým knihovnám funguje pouze na OS Windows. Kompilován byl úspěšně pod MSVS2010 a MSVS2012.

Instrument

K zajištění komunikace s osciloskopem využívá standardu VISA (Virtual Instrument Software Architecture). Námi používaný osciloskop je Agilent DSO-X 3012A, nicméně program by měl fungovat i s jiným osciloskopem, schopným komunikovat pomocí SCPI příkazů přes USB.

Soubory

instrument.h
instrument.cpp

Instrument

Základní konstruktor, pouze vytvoří objekt.

Syntax

```
Instrument()
```

Connect

Zajistí připojení k danému USB instrumentu, tj. k našemu osciloskopu. Pokud se připojení nepodaří, skončí program chybou.

Syntax

```
void Connect(int const instrumentNumber)
```

- `[in] instrumentNumber` – číslo instrumentu. Zjistíme pomocí Agilent Connection Expert[1]

Command

Zašle instrumentu SCPI příkaz, neočekává odpověď.

Syntax

```
void Command (char const *command)sta
```

- `[in] *command` – c string obsahující SCPI příkaz

Příklad

```
#include "instrument.h"

void main() {

	Instrument scope;
	scope.Connect(1);
	scope.Command(":SINGle");
}
```

[1] http://www.home.agilent.com/en/pd-1985909-pn-E2094/io-libraries-suite-162

CommandIEEEBlock

Zašle příkaz ve formě IEEE bloku binárních dat. Lze použít např. Pro nahrání konfigurace osciloskopu, nebo pro vykonání předdefinované sady příkazů.

Syntax

```
int CommandIEEEBlock (char const *command, vector<unsigned char> &ieeeBlock)
```

- `[in] *command` – c string obsahující SCPI příkaz
- `[in] &ieeeBlock` – reference na vektor obsauhující binární data obsahující příkazy k odeslání
- `[out]` vrací počet přenesených bytů

Příklad

```
#include "instrument.h"

void main() {

	vector<unsigned char> rawData;
	/* Do rawData nahrajeme předem uloženou konfiguraci osciloskopu */

	Instrument scope;
	scope.Connect(1);
	scope.CommandIEEEBlock(":SYSTem:SETup", rawData);

}
```

QueryString

Zašle query, tj. příkaz na který očekává odpověď. Odpověď očekává ve formě řetězce ASCII znaků.

Syntax

```
int QueryString (char const *query, vector<char> &response)
```

- `[in] *query` – c string obsahující SCPI query
- `[out] &response` – reference na vektor znaků obsahující odpověď, musí být dostatečně velký aby se do něj vešla celá odpověď. 256 Znaků je VISA standardem požadovaná minimální velikost pro správné fungování.
- `[out]` vrací délku odpovědi v bytech

Příklad

```
#include "instrument.h"

void main() {

	Instrument scope;
	scope.Connect(1);
	/* Get and display the device's *IDN? string. */
	vector<char> idnString(256,0);
	scope.QueryString("*IDN?", idnString);
}
```

QueryDouble

Zašle query, tj. příkaz na který očekává odpověď. Odpověď očekává ve formě jednoho double precision čísla.

Syntax

```
void QueryDouble (char const *query, double &response)
```

- `[in] *query` – c string obsahující SCPI query
- `[out] &response` – reference na double precision číslo

QueryIEEEBlock

Zašle query, tj. příkaz na který očekává odpověď. Odpověď očekává ve formě IEEE bloku, tj. bloku binárních dat se specifickou hlavičkou.

Syntax

```
int QueryIEEEBlock (char const *query, vector<unsigned char> &response)
```

- `[in] *query` – c string obsahující SCPI query
- `[out] &response` – reference na vektor znaků do kterého se zapíše odpověď na query. Vektor je náležitě zvětšen podle velikosti odpovědi.
- `[out]` vrací délku odpovědi v bytech

Příklad

```
#include "instrument.h"

void main() {

	vector<unsigned char> response;
	Instrument scope;
	scope.Connect(1);
	scope.QueryIEEEBlock(":SYSTem:SETup?", response);
}
```

CommandQueryPrint

Zašle příkaz nastavující nějaký parametr, a pak pomocí query ověří hodnotu parametru. Query vytvoří automaticky přidáním otazníku k příkazu. Query a odpověď na něj vypíše na standardní výstup. Zpřehledňuje program v situaci, kdy je potřeba nastavit více parametrů po sobě a kdy je vhodné o jejich nastavení informovat uživatele.

Syntax

```
int CommandQueryPrint(char const *command, char const *commandValue)
```

- `[in] *command` – c string obsahující SCPI příkaz
- `[out] *commandValue` – c string obsahující hodnotu parametru k zapsání
- `[out]` vrací délku odpovědi v bytech

Příklad

```
#include "instrument.h"

void main() {

	Instrument scope;
	scope.Connect(1);
	scope.CommandQueryPrint(":ACQuire:TYPE",          "AVERage");
	scope.CommandQueryPrint(":ACQuire:COUNt",         "2");
	scope.CommandQueryPrint(":TIMebase:MODE",         "MAIN");
	scope.CommandQueryPrint(":WAVeform:UNSigned",     "1");
	scope.CommandQueryPrint(":WAVeform:FORMat",       "BYTE");
}
```

QueryPrint

Zašle query ověřující hodnotu parametru. Query vytvoří automaticky přidáním otazníku k příkazu. Query a odpověď na něj vypíše na standardní výstup. Zpřehledňuje program v situaci, kdy je vhodné o nastavení daného parametru informovat uživatele.

Syntax

```
int QueryPrint(char const *command)
```

- `[in] *command` – c string obsahující SCPI příkaz
- `[out]` vrací délku odpovědi v bytech

Příklad

```
#include "instrument.h"

void main() {

	Instrument scope;
	scope.Connect(1);
	scope.Command		(":SINGle");
	scope.QueryPrint	(":WAVeform:POINts");
}
```

SaveConf

Uloží aktuální konfiguraci osciloskopu do souboru.

Syntax

```
int SaveConf(char const *fileName)
```

- `[in] *fileName` – c string obsahující název souboru, do které ho se bude zapisovat. Soubor je celý přepsán novou konfigurací, pokud existuje. Pokud neexistuje, je vytvořen.
- `[out]` vrací nulu, pokud proběhne zápis správně, tj. pokud bylo přečteno a zapsáno stejně bytů. V případě chyby vrací rozdíl mezi počtem přečtených a zapsaných bytů.

Příklad

Viz. LoadConf

LoadConf

Načte konfiguraci osciloskopu ze souboru. Soubor musí existovat a být ve správném formátu, nedochází k žádné kontrole.

Syntax

```
int LoadConf(char const *fileName)
```

- `[in] *fileName` – c string obsahující název souboru, ze kterého se bude číst. Soubor musí existovat a být ve správném formátu.
- `[out]` vrací nulu, pokud proběhne čtení správně, tj. pokud bylo přečteno a zapsáno stejně bytů. V případě chyby vrací rozdíl mezi velikostí souboru a počtem zapsaných bytů.

Příklad

```
#include "instrument.h"

void main() {

	Instrument scope;
	scope.Connect(1);
	scope.SaveConf("setup.txt");
	scope.LoadConf("setup.txt");
}
```

APDU

Třída, která je pouze strukturou s explicitním konstruktorem. Vytváří buffer pro odesílání APDU case 4 (odesíláme data, očekáváme odpověď) pomocí WinSCard API, pomocí referencí umožňuje jednodušší přístup k polím APDU, jako jsou CLA, INS, P1, P2, Lc a Le. Všechny proměnné jsou stejně jako ve struktuře veřejné.

Soubory

Třída nemá vlastní soubory, je součástí *scard.h*.

Členské proměnné

Jednotlivé členské proměnné jsou inicializovány pomocí inicializačního listu konstruktoru na tyto hodnoty:

```
vector<BYTE> byte;
BYTE &cla = byte[0];
BYTE &ins = byte[1];
BYTE &p1  = byte[2];
BYTE &p2  = byte[3];
BYTE &lc  = byte[4];
BYTE &le  = byte[byte.size()-1];
BYTE * const data = &byte[5];
```

Je tedy možné je adresovat buď pomocí referencí a tedy jména daného pole, nebo pomocí jejich pozice v bufferu.

APDU

Vytvoří APDU case 4, tj. takové které odesílá data a očekává jejich odpověď. Ostatní případy lze jednoduše nasimulovat tím, že se odešle pouze část APDU, případně se zamění funkce pole Lc za Le.

Syntax

```
APDU(int dataSize)
```

- `[in] dataSize` – velikost dat pro odeslání. Např. pro AES je hodnota 16.

Příklad

Viz. Smartcard metoda Send

Smartcard

Zajišťuje komunikaci s kartou přes PC/SC rozhraní, konkrétně pomocí WinSCard API. Je napsán tak, aby komunikoval přes protocol T1, tj. blokově, ne po bytech jako T0. Změna na komunikaci přes T0 je pouhá změna několika parametrů ve volaných WinSCard funkcích.

Kvůli využívání WinAPI jsme se rozhodli používat názvy proměnných používaných ve WinPI, částečně jsme dodržovali i Bulharskou notaci.

Ve zkratce se stará pouze o připojení ke čtečce karet, o připojení ke kartě, případně o znovupřipojení v případě, kdy se spojení a o komunikaci pomocí APDU.

Soubory

scard.h
scard.cpp

Členské proměnné

Jedinou veřejnou členskou proměnnou je *recvBuffer*. Jedná se o vektor, ve kterém je uložena odpověď karty na poslední volání metody *Send.*

Syntax

```
vector<BYTE> recvBuffer
```

Smartcard

Základní konstruktor, pouze vytvoří objekt.

Syntax

```
Smartcard()
```

Connect

Zajistí připojení ke čtečce karet, a k do ní vložené kartě. umožňuje výběr čtečky uživatelem. Komunikuje přes standardní vstup a výstup. Vytvoří kontext pro komunikaci s kartou, je tedy nutné metodu zavolat před jakoukoliv jinou komunikací s kartou. Pokud se spojení s kartou či čtečkou nepodaří, skončí program chybou.

Syntax

```
void Connect()
```

- Varianta, kdy uživatel pomocí standardního vstupu volí čtečku ze seznamu připojených čteček. Seznam je vypsán přes standardní výstup.

Příklad

```
#include "scard.h"

void main() {

	Smartcard card;
	/* Připojení pomocí výběru čtečky uživatelem */
	card.Connect();
}
```

Status

Vyčte z karty ATR[2]. Pokud se vyčtení nepodaří, pokusí se znovu připojit ke kartě. Pokud se znovupřipojení nepodaří, program skončí chybou. Vhodné provést po delší nečinnosti s kartou, neboť může dojít k přerušení spojení. Jednoduchý způsob jak ověřit, že komunikace s kartou je v pořádku.

Na standardní výstup vypíše status karty podle *Smart Card Reader States*[3] a dále vypíše ATR.

Syntax

```
DWORD Status()
```

- `[out]` vrací jednu z hodnot *Smart Card Reader States*

Příklad

```
#include "scard.h"

void main() {

	Smartcard card;
	card.Connect(0);
	card.Status();
}
```

Send

Pošle kartě APDU case 4. APDU musí mít správně vyplněná všechna pole. Odpověď (tj. Response APDU) uloží do členské proměnné `recvBuffer`. Metoda pouze obaluje funkci scardTransmit z WinSCard tak, aby vyhovovala naší struktuře APDU. Pokud je třeba zaslat jiná data, stačí tak učinit jednoduchou úpravou parametrů funkce scardTransmit uvnitř metody Send.

Syntax

```
DWORD Send(APDU const &apdu)
```

- `[in]` `&apdu` – reference na objekt APDU které bude posíláno
- `[out]` vrací hodnotu předanou funkcí WinApi *SCardTransmit*. V případě úspěchu se jedná o kód `SCARD_S_SUCCESS`

Příklad

```
#include "scard.h"

void main() {

	/* Vytvoření APDU */
	APDU apdu(16);
	apdu.cla = 0x80;
	apdu.ins = 0x60;
	apdu.p1  = 0x00;
	apdu.p2  = 0x00;
	apdu.lc  = 0x10;
	apdu.le  = 0x10;
	for (int i=0;i<16;i++) { apdu.data[i] = 0x11; }

	/* Odeslání APDU kartě */
	Smartcard card;
	card.Connect(0);
	card.Status();
	card.Send(apdu);
}
```

---

[2] Answer To Reset

[3] http://msdn.microsoft.com/en-us/library/windows/hardware/ff549009(v=vs.85).aspx

Measurement

Tato třída slouží pro provádění samotného měření a interakci s uživatelem. Její instance se vždy vztahuje ke konkrétnímu připojenému osciloskopu a kartě.

Třída je na rozdíl od předchozích vytvořena přímo za účelem získání dat ke korelační analýze prolamování šifry AES a nepředpokládá se její využití v jiných projektech.

Measurement

Konstruktor pouze pomocí inicializačního listu vytvoří reference na objekty karty a osciloskopu.

Syntax

```
Measurement(Instrument &vi, Smartcard &card)
```

- `[in] &vi` – reference na objekt osciloskopu, se kterým měříme
- `[in] &card` – reference na objekt karty, kterou hodláme měřit

Příklad

```
#include "scard.h"
#include "instrument.h"
#include "measurement.h"

void main() {

	Instrument scope;
	Smartcard card;
	Measurement measurement(scope, card);
}
```

MainMenu

Hlavní rozcestník pro uživatele, zobrazí menu s možnými akcemi, tj. s možnými metodami třídy Measurement, které může uživatel zavolat. Komunikuje textově přes standardní vstup/výstup.

Ostatní metody lze volat explicitně, nicméně předpokládá se explicitní volání pouzeMainMenu, ze které jsou později volány všechny ostatní metody třídy Measurement.

Syntax

```
void MainMenu ()
```

PrintScreen

Vytvoří snímek obrazovky osciloskopu a uloží jej do souboru *printscreen.png*. Název souboru lze změnit v kódu metody.

Syntax

```
void PrintScreen ()
```

LoadSetupFile

Načte nastavení osciloskopu ze souboru. Jméno souboru zadává uživatel přes standardní vstup. Soubor musí existovat a být ve správném formátu, neprobíhá kontrola obsahu.

Syntax

```
void LoadSetupFile ()
```

SaveSetupFile

Uloží nastavení osciloskopu do souboru. Jméno souboru zadává uživatel přes standardní vstup. Soubor nemusí existovat, a pokud existuje, je přepsán.

Syntax

```
void SaveSetupFile ()
```

TestRun

Testovací běh šifrování na kartě, kdy neprobíhá měření. Kartě je odesíláno v cyklu APDU s příkazem šifrování a se vzorovými daty. Data jsou odesílána v nekonečném cyklu, který je ukončen stiskem libovolné klávesy.

Metoda má pomoci správnému nastavení zobrazování dat na osciloskopu. Jakmile zobrazení odpovídá požadavkům, je dále možné konfiguraci uložit pomocí metody *LoadConf*.

Důležité: Přednastavený tvar APDU je potřeba v kódu upravit tak, aby odpovídal naší kartě.

Syntax

```
void TestRun ()
```

MeasurementRandom

Samotná funkce použitá pro získání průběhů spotřeby energie kartou při šifrování. Uživatel je po zavolání metody přes standardní vstup vyzván k zadání počtu měření, která se mají provést.

Kartě jsou odesílány náhodně generované plaintexty (s pevně daným seedem pro lepší reprodukovatelnost měření) a vyčítány jsou příslušné ciphertexty, které jsou ukládány do souborů *plaintext.txt* a *ciphertext.txt* respektive. Průběhy napětí jsou ukládány do souboru *traces.bin* v binární formě.

V případě, že nám nevyhovuje způsob generování odesílaných dat, formátu výstupu, či způsobu měření, je třeba upravit kód metody, či vytvořit novou metodu z této vycházející.

Důležité: Po každém zavolání metody jsou soubory *plaintext.txt, ciphertext.txt* a *traces.bin* přepsány.

Důležité: Přednastavený tvar APDU je potřeba v kódu upravit tak, aby odpovídal naší kartě.

Syntax

```
void MeasurementRandom ()
```

GetPreamble

Metoda která z osciloskopu vyčte hodnoty preambule posledního provedeného měření. Hodnoty uloží do privátních členských proměnných třídy Measurement.

Momentálně nevyužito, pro případné budoucí využití.

Syntax

```
void GetPreamble ()
```

MeasurementCSV

Momentálně nevyužito, ponecháno pouze jako demonstrace využití hodnot preambule měření pro lepší reprezentaci výstupních dat.

Syntax

```
void MeasurementCSV ()
```