

SmartTrust WIB 1.3

R&D Mobile Communications – June 2004

Julien SIMON
j.simon@oberthurcs.com

The one-slide story

The WIB is (mostly)
a WAP browser,
running on a SIM Toolkit card.

WIB = Wireless Internet Browser
WAP = Wireless Application Protocol

Outline

1. WAP overview

2. WIB & WIG

3. WIB commands

- WML
- SIM Toolkit commands
- WIB-specific commands

4. Local Scripts

- Local storage
- Event & menu management

5. Plug-in management

6. Plug-ins

- General-purpose plug-ins
- Secret key plug-ins
- Public key plug-ins
- Call Control applet

7. Conclusion

8. Appendices

- WML
- PKCS

Wireless Application Protocol

- WAP 1.0 was released in 1998 and evolved into 1.3.
- WAP 1.x doesn't support standard Internet protocols and languages : a **WAP gateway** is required.

WAE: Application Environment

WSP: Session Protocol

WTP: Transaction Protocol

WTLS: Transaction Layer Security

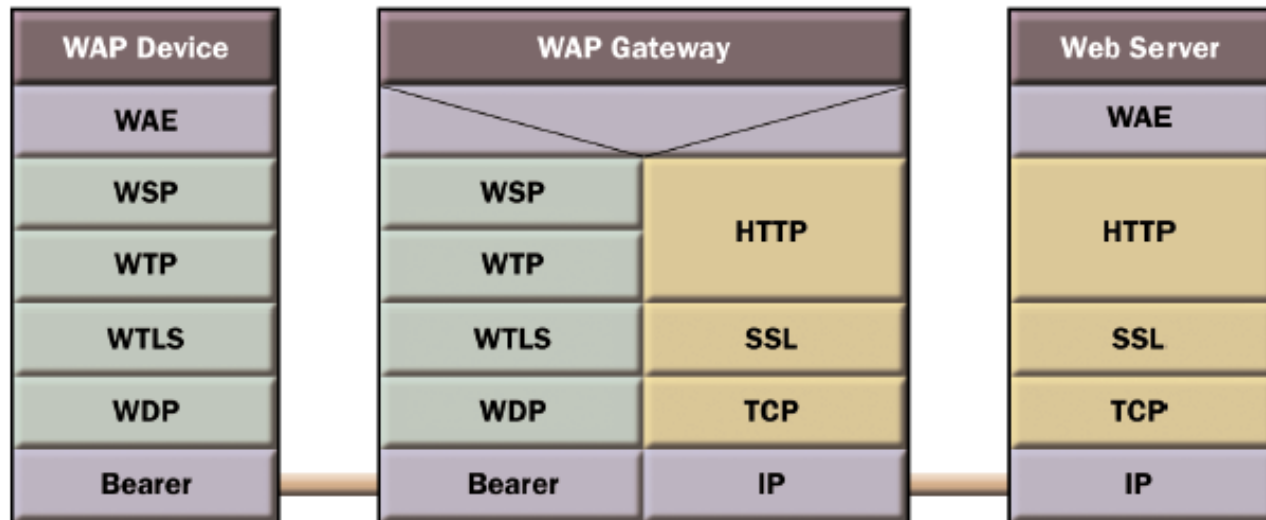
WDP: Datagram Protocol

HTTP: Hyper Text Transfer Protocol

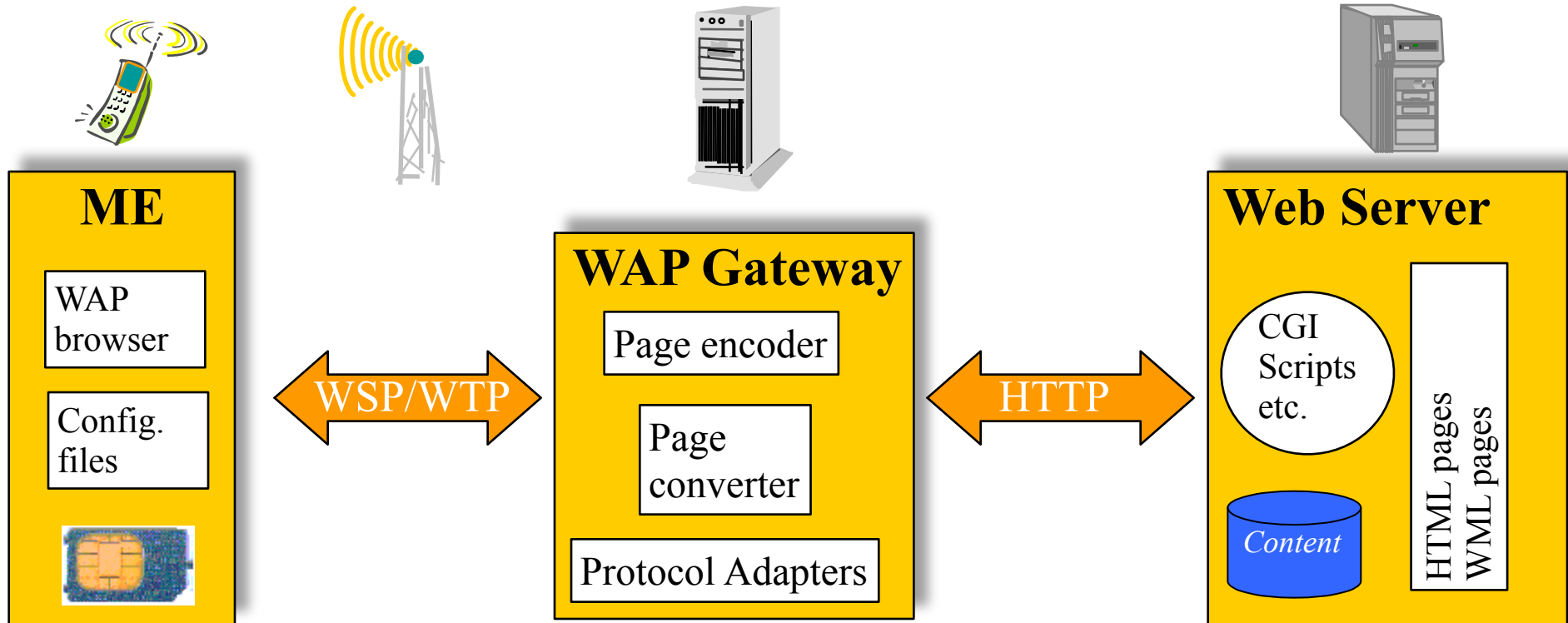
SSL: Secure Sockets Layer

TCP: Transmission Control Protocol

IP: Internet Protocol



WAP Architecture



ME = Mobile Equipment

HTTP = Hyper Text Transfer Protocol

WSP = Wireless Session Protocol

WTP = Wireless Transaction Protocol

WML = Wireless Markup Language

HTML = Hyper Text Markup Language

CGI = Common Gateway Interface

Telephony features are voluntarily omitted.

Wireless Markup Language

- The WAE supports the **Wireless Markup Language** (WML).
 - ◆ It is similar to (but not a subset of) the Hyper Text Markup Language (HTML).
 - ◆ WML is more dynamic than HTML: variables, etc.
- The equivalent of an HTML page is a **WML deck** (transfer unit) which contains one or more **WML cards** (display unit).
- Because of the very limited bandwidth of current networks, WML is translated into a very compact binary representation, called **bytecode**.
 - ◆ This bytecode has nothing to do with Java Card bytecode !
 - ◆ A WAP device must be bytecode-aware in order to decode incoming decks and encode outgoing messages.

From WML to menu

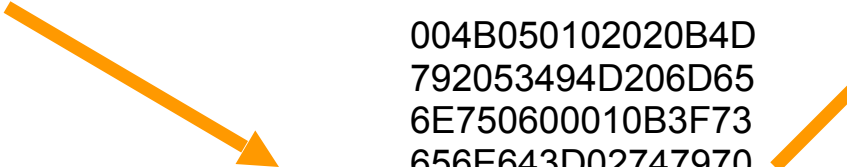
```

<WML>
  <CARD>
    My SIM menu
    <DO TYPE="ACCEPT" LABEL="Next">
      <GO URL="#card2"/>
    </DO>
  </CARD>

  <CARD NAME="card2">
    <DO TYPE="ACCEPT">
      <GO URL="?send=$type"/>
    </DO>
    Services
    <SELECT KEY="type">
      <OPTION VALUE="em">Email</OPTION>
      <OPTION VALUE="ph">Phone</OPTION>
      <OPTION VALUE="fx">Fax</OPTION>
    </SELECT>
  </CARD>
</WML>
  
```

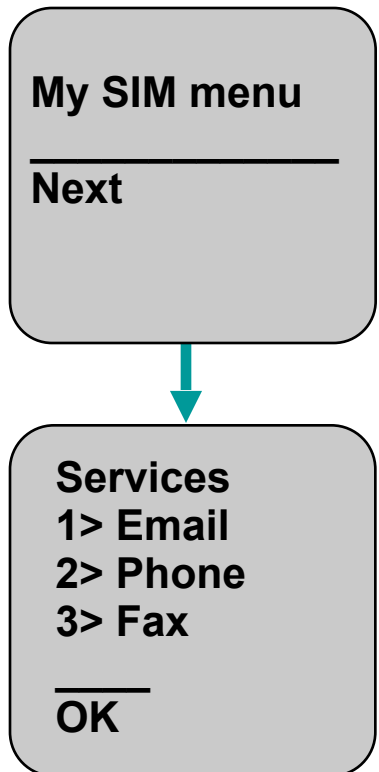
WML : 473 bytes

bytecode : 77 bytes (-83%)

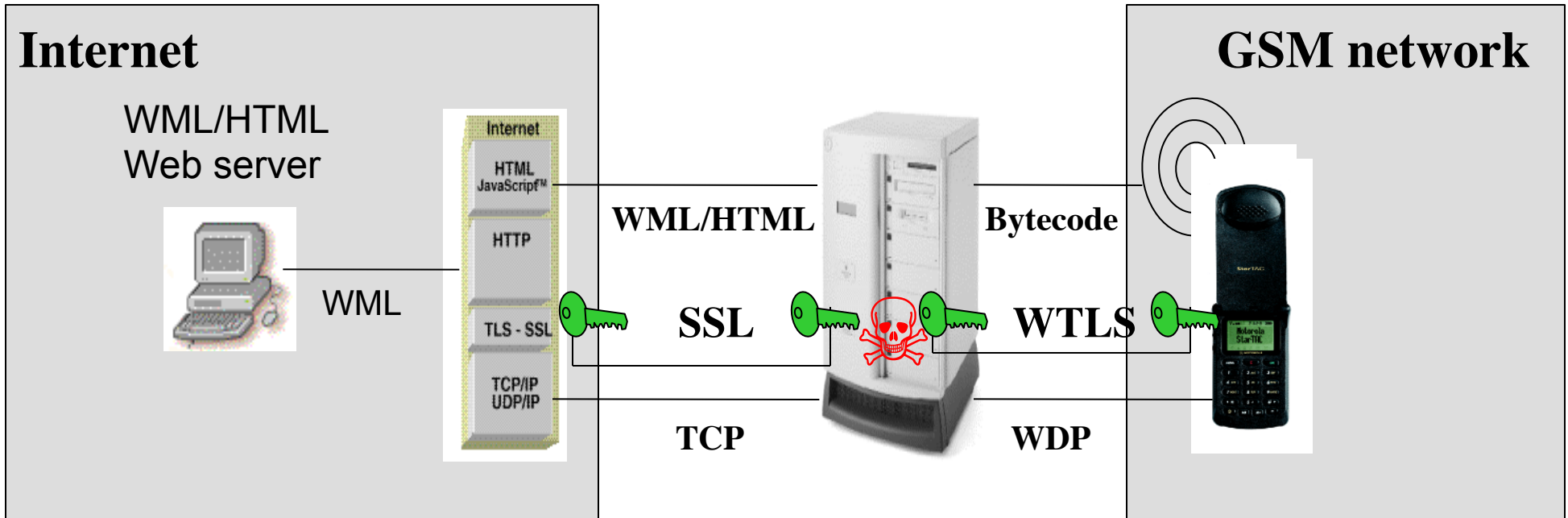


```

004B050102020B4D
792053494D206D65
6E750600010B3F73
656E643D02747970
65042601085365727
66963657305456D61
696C02656D000550
686F6E65027068000
34661780266780006
000600
  
```



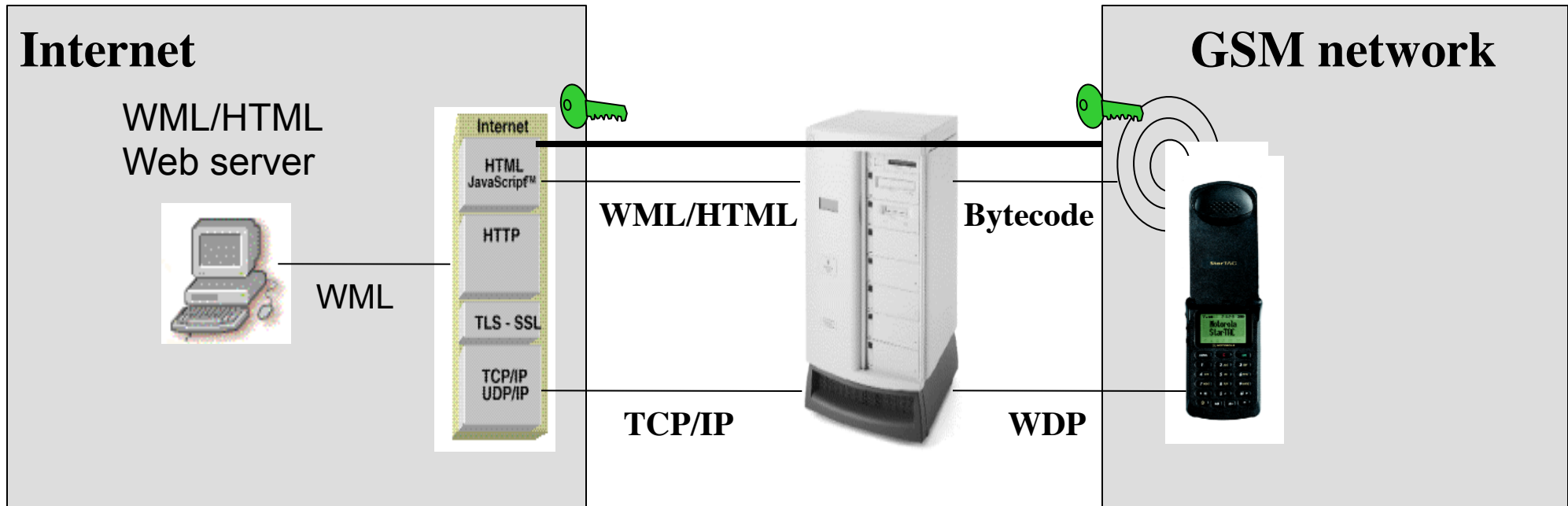
WAP (in)security



There is no end-to-end security, because of the WTLS/SSL gateway. This is the infamous **“WAP gap”** problem.

Can you **really** trust the operator and the security of its network ?

End-to-end security at application level

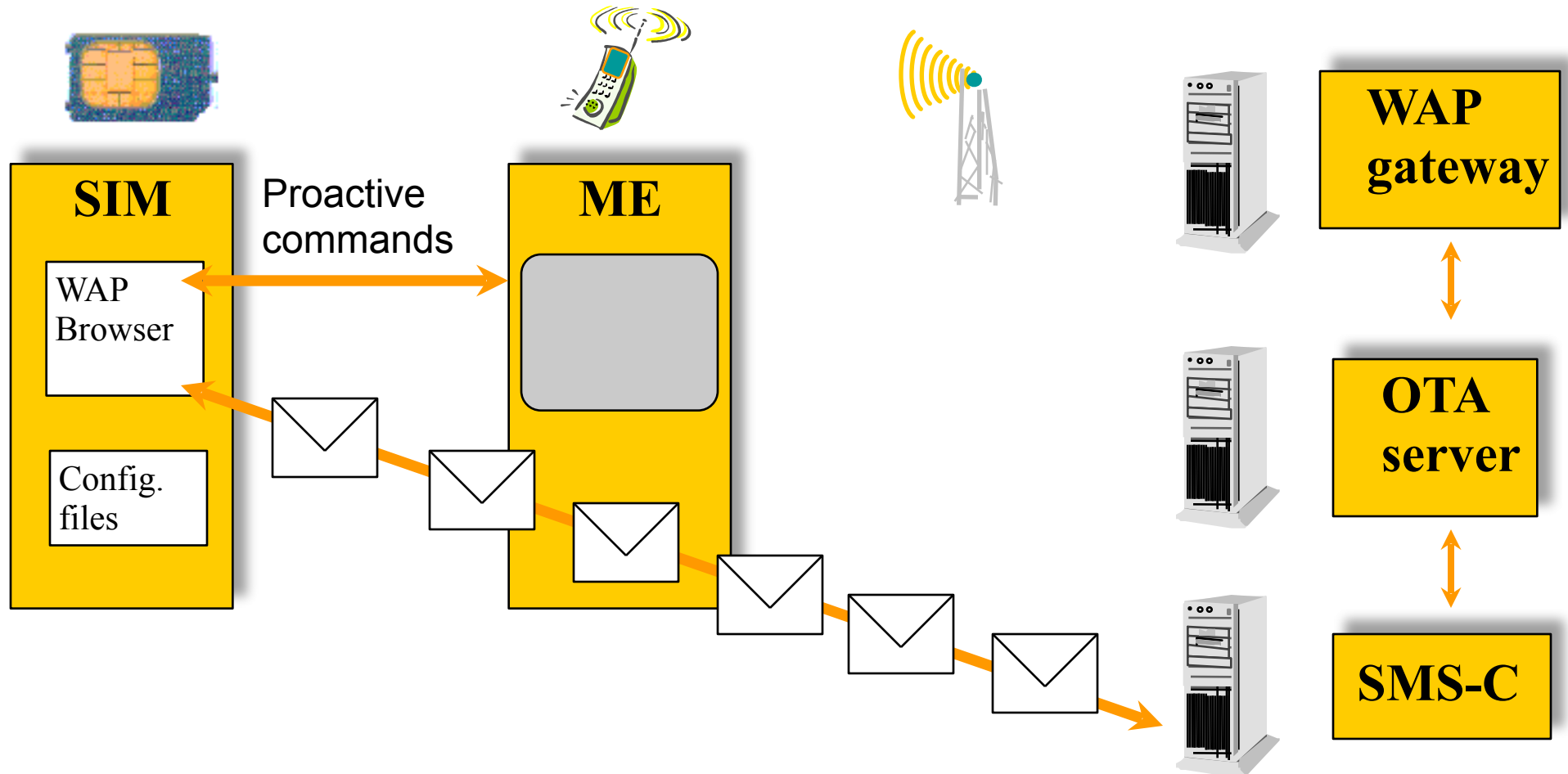


Data is protected by the application,
i.e. the bytecode is encrypted/decrypted by the **WAP browser** (or
by an add-on) with user-level keys.

Move over, browser !

- Moving the browser to the smartcard solves several issues.
 1. There is a HUGE installed base of cheap Phase 2+ phones
→ users can use WAP now and give more money to the operator ! As long as their phone supports what the WIB needs...
 2. Setting up WAP on the phone is a nightmare for the average human : thus, no one uses it and the operator doesn't make any money
→ WIB set-up is pre-loaded at personalization and may be modified OTA.
 3. Different phones have different browsers : the operator gets a subset of the standard... and a superset of browser bugs
→ a unique browser on all cards (no matter what the phone is).
 4. A smartcard offers better security (key storage, state-of-the art crypto, etc.).

WAP-enabled smartcard



The Invasion of the Killer Browser

- The services offered by a SIM Toolkit applet are defined by its implementation and personalization.
- Adding new services means replacing the applet with a newer version, which usually means replacing the SIM card...
- The same services can be provided using **local pages**.
- The browser never changes : only the local pages are updated.
- Additional features may be added OTA using **plug-ins** (PKI, etc).
- The user may **pull** content from the network (Portal, WWW).
- The operator may **push** any kind of targeted content to the user.

On second thought...

- Some specific operator requirements cannot be – and will never be – met by a generic browser... unless you develop a specific plug-in for each of them. This defeats the purpose, doesn't it ?
- If the operator doesn't have any specific requirement, a hand-crafted SIM Toolkit Applet will always be smaller and faster than a generic (read: bloated) browser.
- Deploying a browser has major impacts on the operator network : much more SMS traffic, WAP gateway, etc.
- **But hey, let marketing worry about that, right ?**

A Band Of Browsers

- SmartTrust **Wireless Internet Browser**.
- SIM Alliance **S@T browser**.
- 3GPP **USAT Interpreter** :
 - ◆ 21.112 & 31.11{2,3,4} Release 5 : architecture, protocol, core browser.
 - ◆ 31.113 Release 6 : PKI plug-ins.
- And let's not forget proprietary browsers like the PICO.
- These browsers all support the WML standard (or at least a large subset of it) as well as proprietary extensions (**S@TML**, **SmartTrust WML**, etc). This is similar to the Netscape/Internet Explorer situation.

So... what's the big deal ?

- A WAP browser really belongs on the phone, where processing power and memory are plentiful.
- A Java smartcard is just about the worst place to implement a WAP browser.
 - ◆ Minimal resources : **8-bit processor**, **128 bytes of stack**, around **1 Ko of RAM storage**, **slow EEPROM writes**.
 - ◆ Cumbersome transport layer (**03.40 / 03.48**) : **small PDU size**, **high latency**.
 - ◆ Language / API issues.
- Face it : the WIB is a Virtual Machine on top of the Java Card Virtual Machine...

SmartTrust

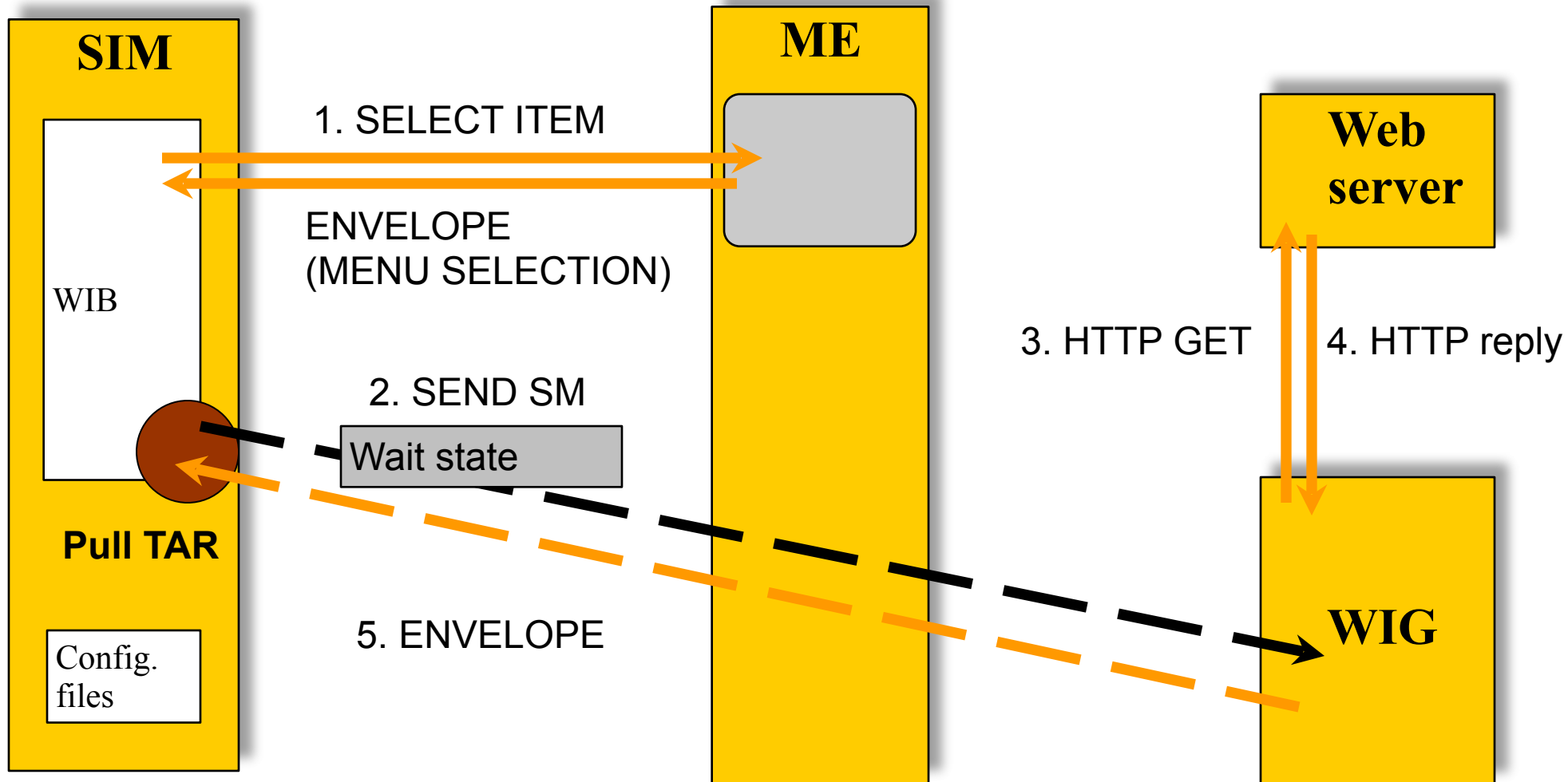
« SmartTrust is a leading developer of infrastructure solutions designed to provide mobile subscribers with easy to use and secure personalized services. »

- SmartTrust maintains the WIB specification, which is implemented by a number of smartcard vendors.
- SmartTrust provides server software :
 - a WAP gateway for the WIB, called **Wireless Internet Gateway** (WIG).
 - an OTA management platform, called **Delivery Platform** (DP).

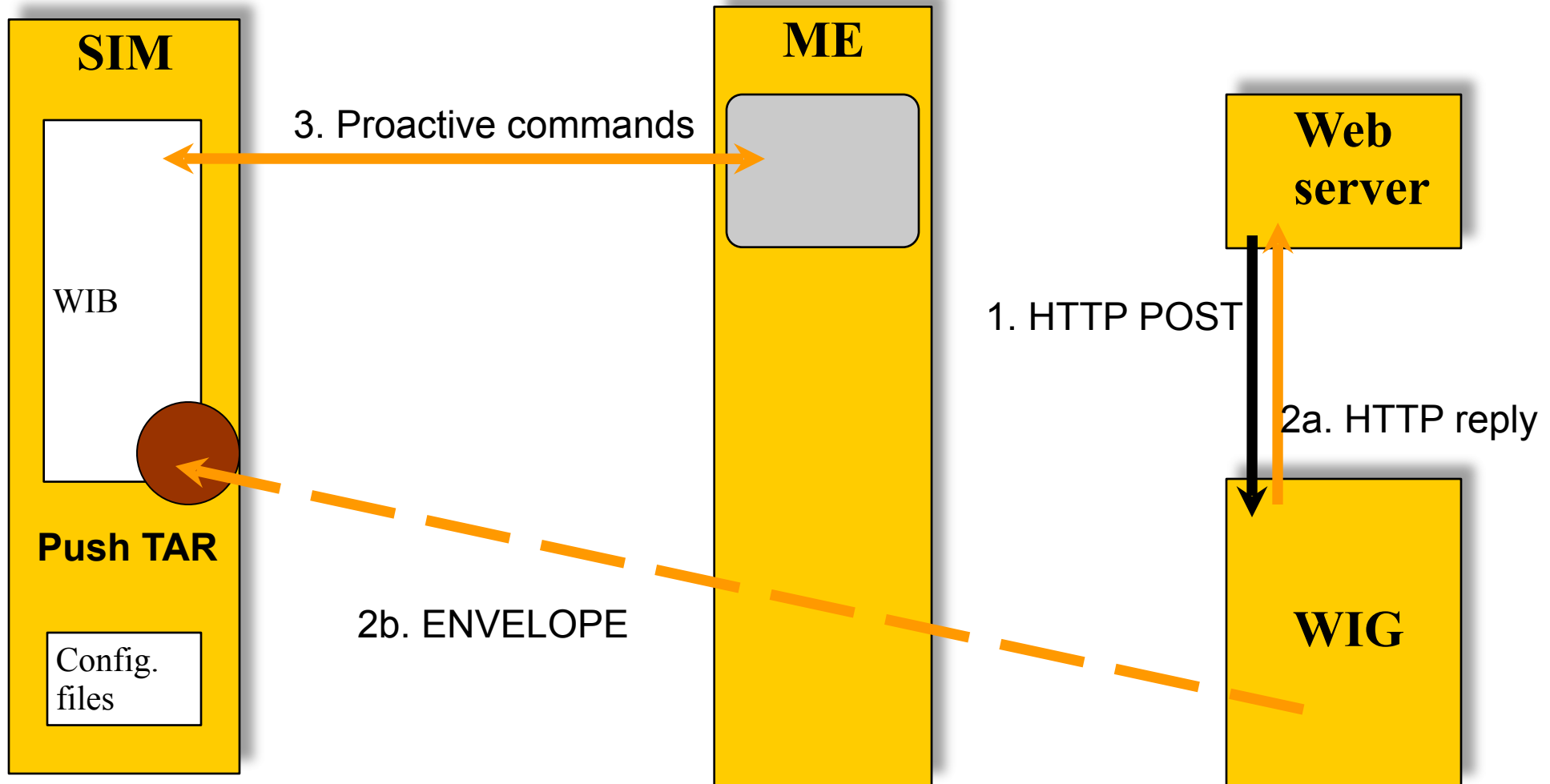
WIB-WIG Communication

- WIB and WIG talk to each other using the Short Message Service (SMS).
- **Concatenated messages** may be used in both directions.
- The WIB uses **3 Target Application Reference (TAR)** values.
 - ◆ Pull TAR : WIB-initiated messages.
 - ◆ Push TAR : WIG-initiated messages.
 - ◆ Administrative TAR : WIG-initiated administrative messages.
 - ◆ 1 extra TAR for Vodafone : the Data Download TAR, which manages the “OTA Script Executed” event.
- All messages have a **03.48 security header**, with one of these security levels:
 - ◆ L0 : no security.
 - ◆ L1 : authentication counter + 3DES MAC.
 - ◆ L2 : authentication counter + CRC-32 MAC + 3DES encryption.
 - ◆ L3 : authentication counter + ISO9797 MAC + 3DES encryption.
- The 03.48 Proof Of Receipt (PoR) mechanism is not used : all messages are commands.
- In the future, other bearers than SMS may be used.

Pull Mode



Push Mode

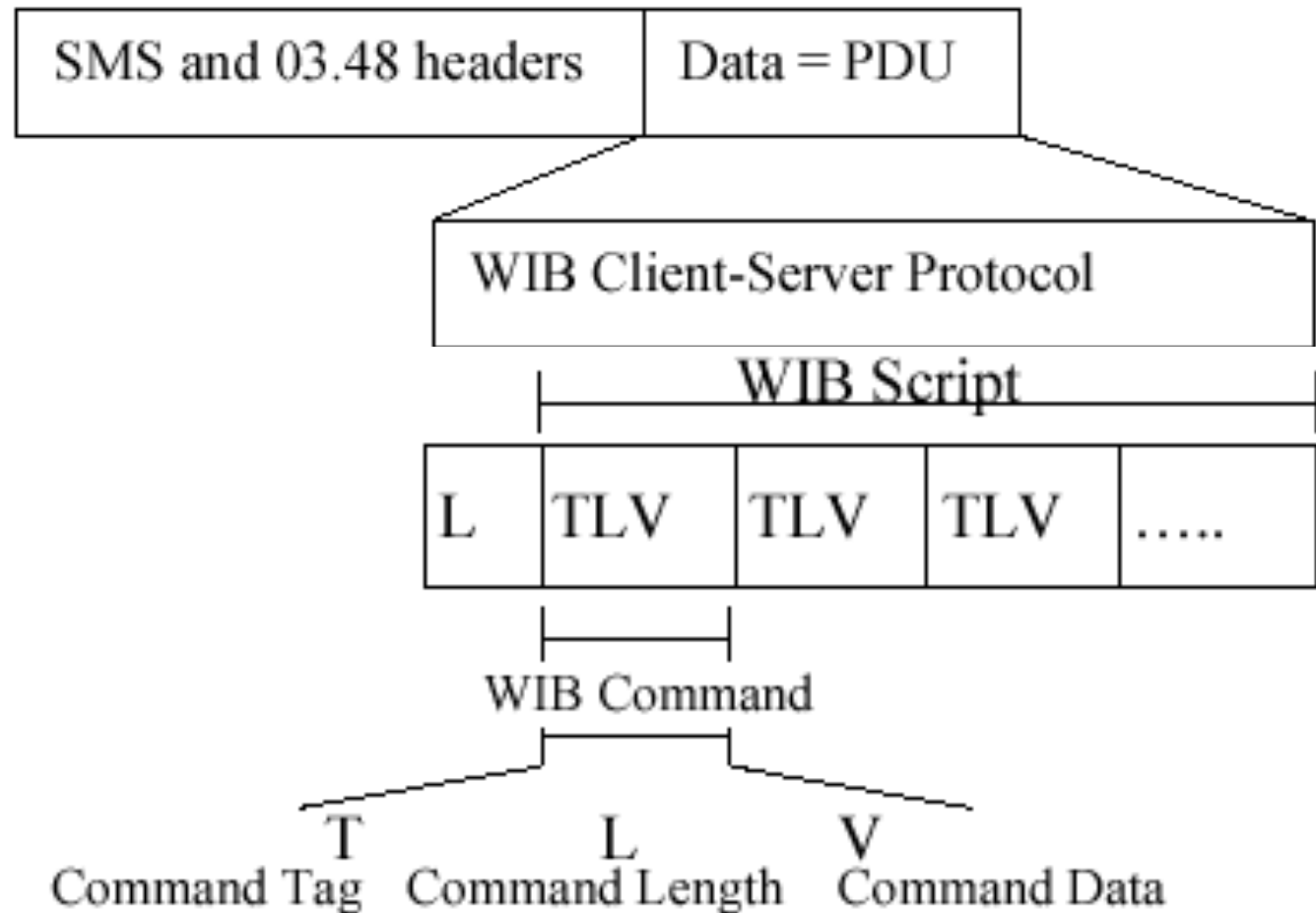


WIB Script

A **WIB script** is a sequence of **WIB commands**.

A script is run in order until :

- the last command is reached.
- the Exit command is run.
- the Cancel button is pressed.
- the Wait state is entered.



WIB commands

WML

SIM Application Toolkit commands

WIB Commands

- The WIB supports a subset of **standard WML elements** (see Appendix A) :
 - ◆ Deck : `<wml>...</wml>`
 - ◆ Card : `<card>...</ card>`
 - ◆ Paragraph (unit of displayable text) : `<p>...</ p>`
 - ◆ Text formatting : `
`, `<i>...</i>`, `...`, `<u>...</u>`, etc.
 - ◆ Variables : `<setvar/>`
 - ◆ User input : `<input/>`
 - ◆ User selection : `<select/>`, `<option/>`
 - ◆ User Navigation : `<do>...</do>`, `<go/>`, `<a>...`
- All these commands are bytecode-encoded into TLVs.

SIM Application Toolkit commands

- The WIB supports the SIM Application Toolkit (SAT) commands.
 - ◆ Some of the commands may be called explicitly in a WML page using the **go href** element.
 - ◆ Although the **go** element is used, no message is sent to the server : the command is executed locally on the SIM.
 - ◆ Supported SAT Commands depend on the mobile.
- The list of available commands is : Display Text, Get Input, Launch Browser, Provide Local Information, Play Tone, Refresh, Select Item, Send Short Message, Send USSD, Set Idle Mode Text, Set Up Call, Timer Management.
- All these commands are bytecode-encoded into TLVs.

SAT : Send Short Message

A text SM is sent to MSISDN "0706754321", using the specified Service Center "+46705008999" (regardless of the default value in the mobile station).

```
<card>
```

```
<p>
```

```
  <do type="vnd.smarttrust.extended">
```

```
    <go href="http://server#wigSendSM('Hello! ', , ,  
      '0706754321', '+46705008999')"/>
```

```
  </do>
```

```
</p>
```

```
</card>
```


SAT : Set Up Call

If the ME is not busy with another call, "0706754321" is called.
No text is displayed and no automatic retry will be made.

```
<card>
```

```
<p>
```

```
<do type="vnd.smarttrust.extended">
```

```
<go href="http://server#wigSetupCall(00,,,,,'0706754321')"/>
```

```
</do>
```

```
</p>
```

```
</card>
```

SAT : Error Management

- Because of ME diversity and application complexity, Error Management is an important part of the WIB.
- For all supported SIM Toolkit commands, a behavior and/or an **error code** are defined for each TERMINAL RESPONSE value.
- EF_ERROR_TEXT (6F02) stores the error message to be displayed for a given error code.
 - ◆ If an error message contains the %D wildcard, a detailed error report will be displayed :
Error code, number of the WIB command, Terminal Response value / file identifier.
 - ◆ Error code 0xDF is generic : it's the only mandatory entry in EF_ERROR_TEXT.

WIB-specific commands

- The WIB supports **specific commands**, which may originate from :
 1. Explicit WML calls, using the **go href** element.
 2. WML processing by the WIG.
- The list of commands is :
 - ◆ Check Terminal Profile : check if specified bits are set in the Terminal Profile and jump.
 - ◆ Branch On Variable Value : relative jump (variable-based).
 - ◆ **Execute Local Script** : run a script stored on the SIM.
 - ◆ Exit : terminate the current script.
 - ◆ New Context : clear the variable table, partially or totally.
 - ◆ **Plug-in** : invoke a plug-in
 - ◆ Set Return Tar Value : set the destination of the next message submitted to the server.
 - ◆ Set Script Buffer Size To Variable.
 - ◆ Set Version Information to Variable : store WIB version and plug-in information in a variable.
 - ◆ Skip : direct relative jump.
 - ◆ Variable madness: Substring, Add/Subtract, Convert (HEX-BCD-base 10), Group/Ungroup, Swap Nibbles, GSM-UCS2 conversion.
- All these commands are bytecode-encoded into TLVs.

WIB administrative commands

- WIB 1.3 supports **administrative commands**, which deal either with Local Scripts or with plug-ins.
- They must be pushed to the **Administrative TAR**.
- **Get/Set Script Trigger Mode** : event and menu management.
- **Get Menu** : retrieve the current menu.
- **Get Script Info** : retrieve the identifier and address of one or several Local Scripts.
- **Install/Remove plug-in** : activate/deactivate a plug-in.
- All these commands are bytecode-encoded into TLVs.



Local Scripts

Local Storage

Script Addressing

Event management

Menu management

Local Scripts

- A WIB script may also be stored locally, i.e. “pre-loaded” on the SIM in EF_Bytecode (6F03).
- They may be used to provide “applet-like” features.
- The Execution of a **Local Script** may be triggered by :
 1. An **Execute Local Script** WIB command.
 2. The selection of a **menu entry** associated to a Local Script.
 3. The occurrence of a **SIM Toolkit event** associated to a Local Script.
 4. The expiration of a **Timer** associated to a Local Script using the Timer Management SAT command.

Local Scripts and Menus

- When the SIM card is activated, the WIB collects information from EF_Menu (6F18) and EF_Menu_Title (6F1E) to build its menu.
- A menu entry is made of :
 - ◆ An item status,
 - ◆ A item identifier,
 - ◆ A menu ordinal,
 - ◆ An icon identifier,
 - ◆ A menu text.
- The item identifier points to a **Local Script**.
- Administrative commands : **Get Menu, Get/Set Script Trigger Mode, Get Script Info**

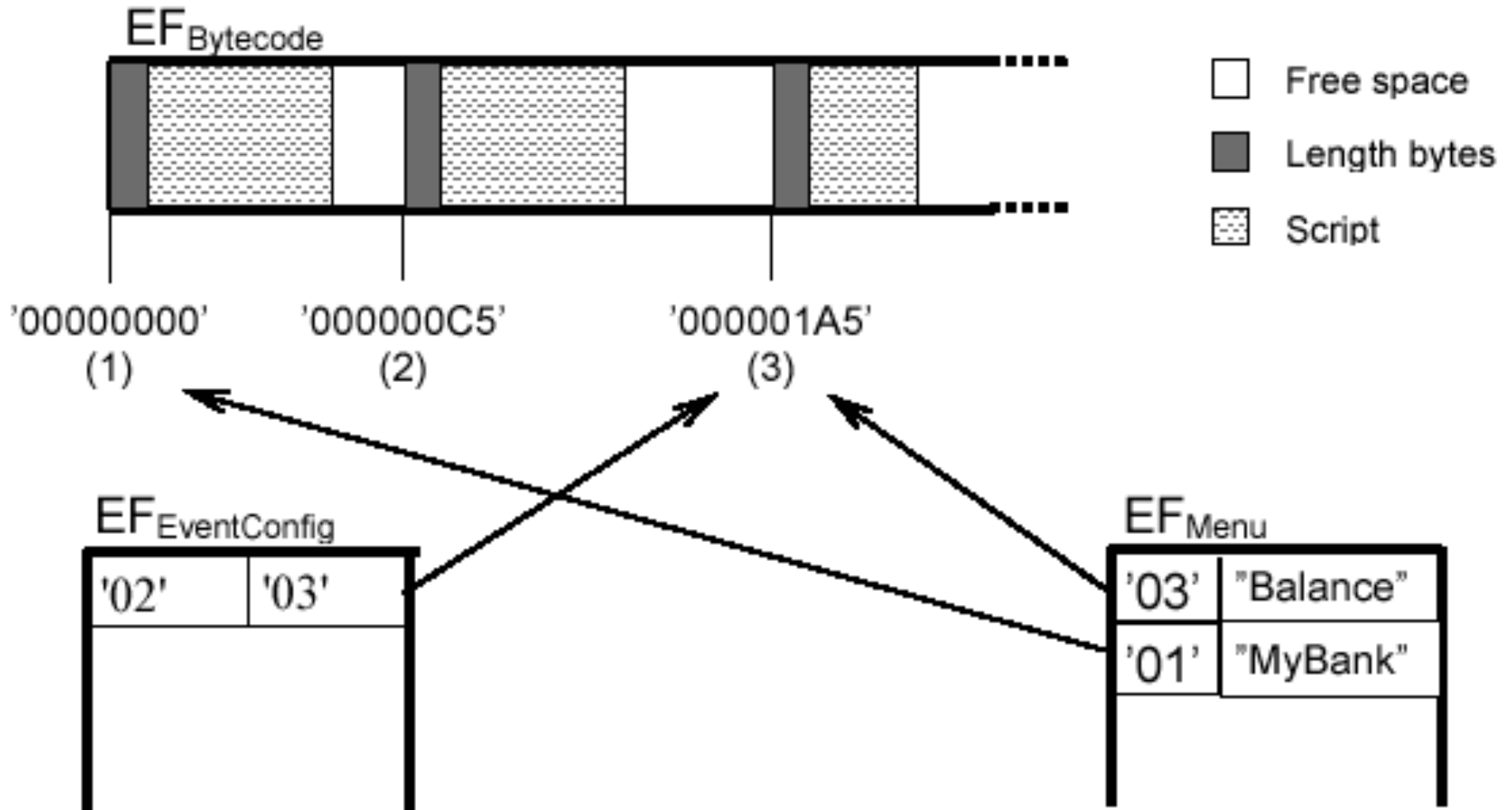
Local Scripts and Events

- EF_EVENT_STARTUP (6F0B) lists the **SIM Toolkit events** the WIB should listen to. A **Local Script** identifier is bound to each event.
- At startup, the WIB reads EF_EVENT_STARTUP and registers the events which are supported by the ME (according to TERMINAL PROFILE).
- When one of these events occurs, the matching **Local Script** is executed.
 - ◆ If the WIB is already busy (or if the Wait For Response state is active), the event is **queued**. If an event of the same type is already queued, the new one overrides it.
 - ◆ Queued events are processed according to their **priority**, i.e. the order in which they appear in EF_EVENT_STARTUP.
- Administrative commands : **Get/Set Script Trigger Mode, Get Script Info**

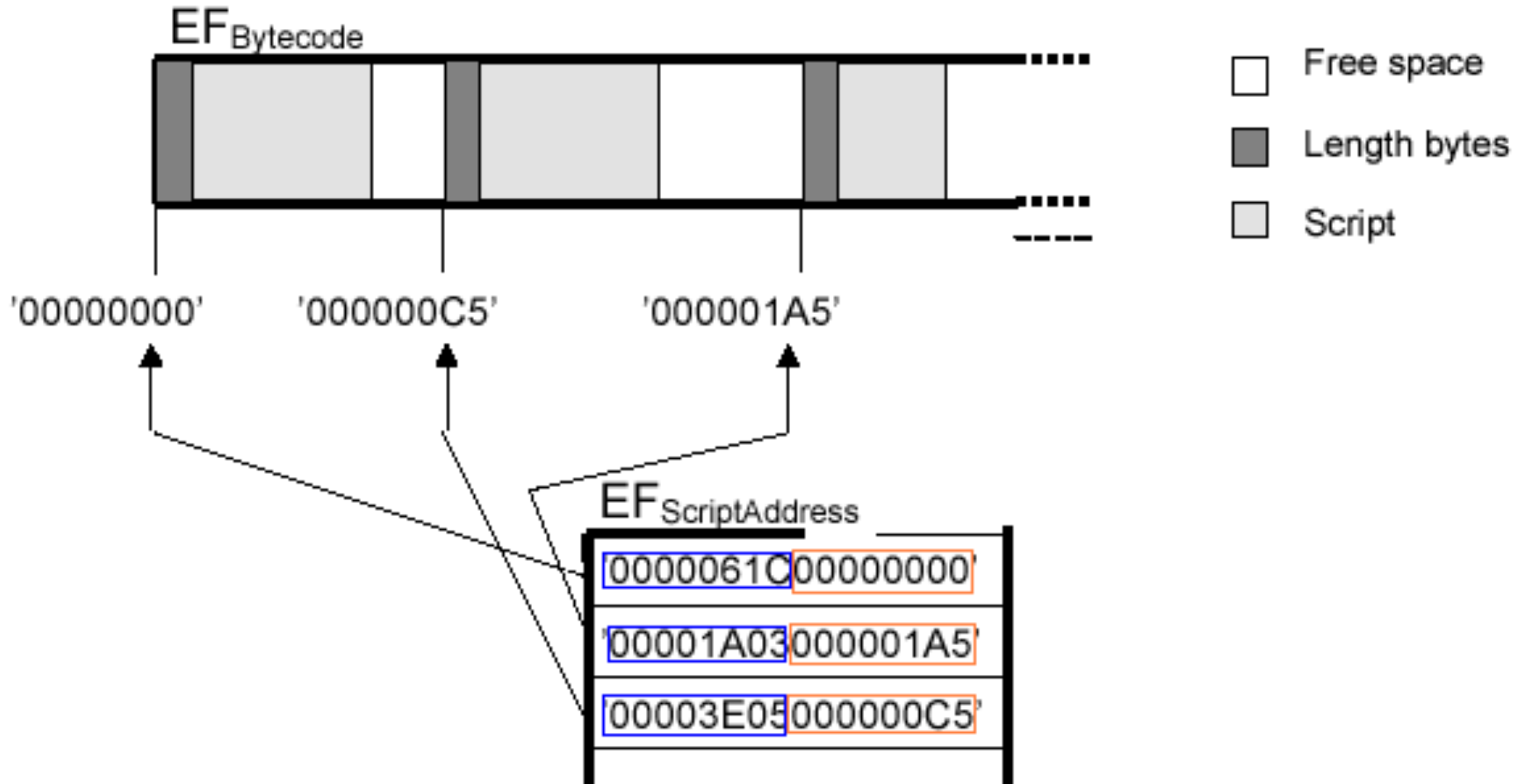
Script Addressing

- Script Addressing is the conversion of a script identifier into a bytecode “pointer” in EF_Bytecode.
- The WIB may be configured in **Absolute Addressing** mode or in **Relative Addressing** mode.
 - ◆ The addressing mode is separately configurable for menu selection and events.
 - ◆ The Timer Management and Execute Local Script commands always use the script identifier listed in EF_SCRIPT_ADDRESS (6F1D).
- Script identifiers and addresses may be retrieved using the **Get Script Info** WIB administrative command.

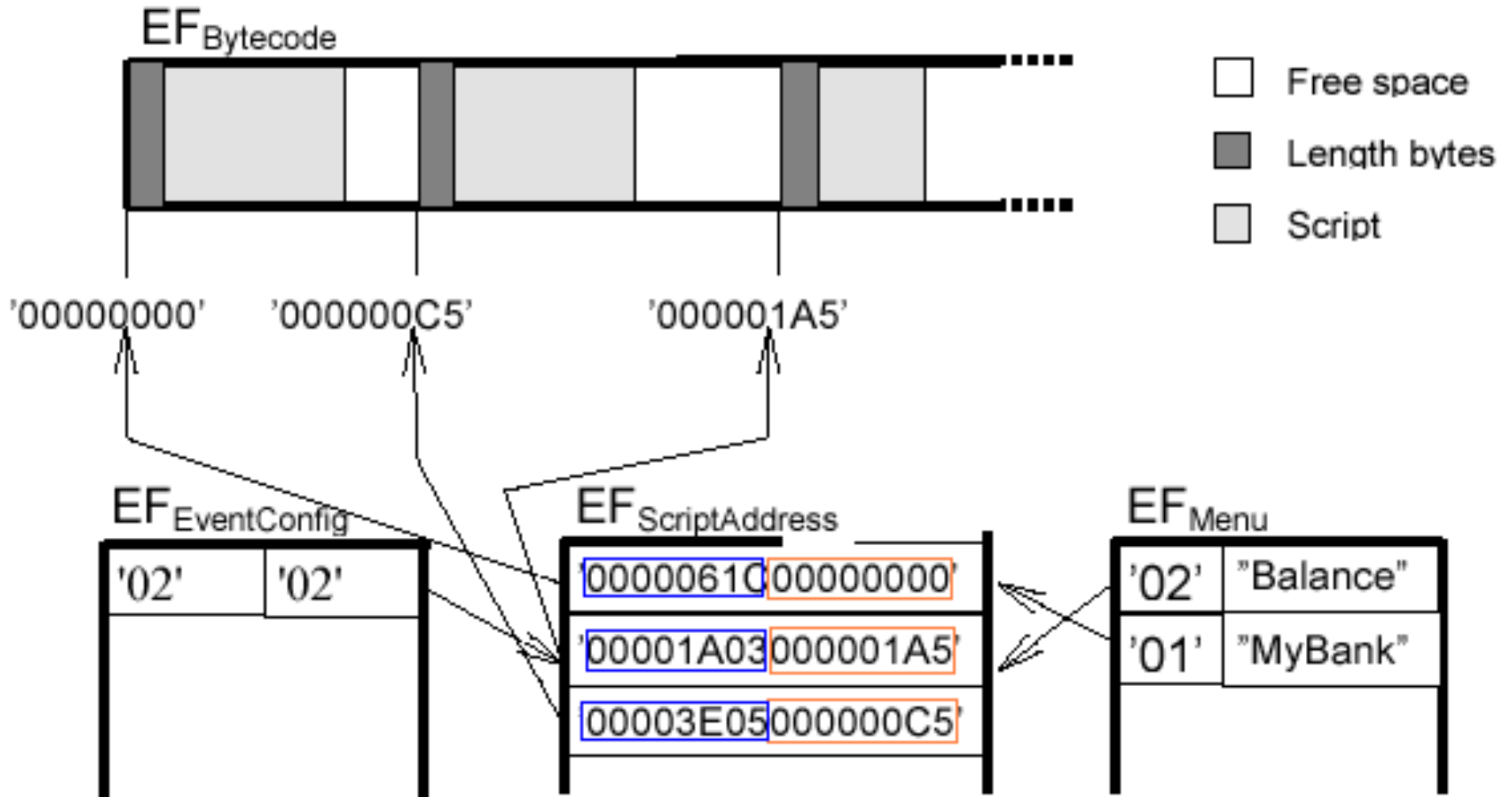
Relative Script Addressing



Absolute Addressing (1/2)



Absolute Addressing (2/2)



Plug-ins

Plug-in management

General purpose plug-ins

Cryptographic plug-ins

Plug-in architecture

- A plug-in is a normal SIM Toolkit applet.
 - ◆ It extends the WIB by adding new features :
 - *Data storage,*
 - *Cryptography : the WAP gap must be closed !*
 - ◆ It may be installed at personalization time.
 - ◆ Plug-ins are usually small enough to be installed OTA.
- A plug-in **registers** to the WIB using a **Shareable** interface.
 - ◆ Registration takes place on Terminal Profile and Status events.
 - ◆ A plug-in is identified by a **text name**.
- When the WIB receives the **Plug-In** command :
 - ◆ it uses the plug-in name to locate the appropriate applet.
 - ◆ It invokes the plug-in using another **Shareable** interface.
- The Shareable interfaces are not defined by SmartTrust.

Plug-In invocation

- This example invokes the PKCS#1 plug-in through the Plug-In WIB command :

```
<setvar name="CES" value="&#x02;" />
```

```
<setvar name="TTBS" value="&#x54;&#x65;&#x78;&#x74;" />
```

```
<go href="http://server#wigObject('P1', '$(CES)$(TTBS)', 'STEXT'  
  ')" />
```

- **P1** : plug-in name.
- **CES** : character-encoding scheme (02 = GSM).
- **TTBS** : text to be signed.
- **STEXT** : signature.

Plug-in administrative commands

- Install Plug-in and Remove Plug-in have misleading names.
- **They do not deal with applet installation and removal** : this is still performed using the Global Platform commands (INSTALL LOAD, INSTALL INSTALL, DELETE).
- **These commands deal with plug-in activation/deactivation**, i.e. whether an registered plug-in may be invoked by the WIB or not.

The (in)famous WIB plug-ins

- General-purpose plug-ins
 - ◆ **Persistent Variables** (Vodafone-specific) : data storage.
 - ◆ **User Data** : secure data storage & display.
- Cryptographic plug-ins
 - ◆ **Standard Security, Universal 3DES Signature** : 3DES encryption/decryption/signature.
 - ◆ **PIN Management** : key access condition management.
 - ◆ **PKCS#1, PKCS#7, Fingerprint** : RSA signature.
 - ◆ **Asymmetric Decryption** : RSA decryption.
 - ◆ **Public Key Retrieval** (Vodafone-specific).
- Call Control applet
 - ◆ not a plug-in, but while we're on the subject...

Persistent Storage (VPS)

- This is a **Vodafone** plug-in (Vodafone Generic 2 specification).
- The WIB cannot store persistent information from one script to another.
- VPS allows the WIG to :
 - ◆ store data into a pre-defined transparent file (EF_VPS : 0x6000).
 - ◆ retrieve data from EF_VPS (trailing 0xFF's not ignored).
 - ◆ retrieve text from EF_VPS (trailing 0xFF's ignored).
- It is intended for Vodafone use only and will not be used to store user sensitive data such as credit card numbers.

User Data (*UD*)

- These plug-ins are used to handle user-specific information (**User Data Objects**) in a secure way.
 - ◆ A User Data object is a TLV, containing a value and an alpha-identifier.
 - ◆ The operator creates the data, which is stored in the EF_USER_DATA_OBJECTS file (6F61).
 - ◆ The user and the operator maintain the data.
 - ◆ Data access is protected by a PIN.
- **Display User Data** (*DUDA*) : display and update the value of a Data Object.
- **Encrypted User Data** (*EUDA*) : retrieve and encrypt (3DES) the value of Data Objects.

Standard Security Plug-ins

- **3DES CBC encryption**, without any user interaction (*ENCR*)
 - ◆ Input data : key index + unpadded data.
 - ◆ Input data is then zero-padded by the plug-in (0 to 7 bytes).
 - ◆ Output data : padded ciphered data, prefixed by a byte set to the padding length.
- **3DES CBC decryption**, without any user interaction (*DECR*)
 - ◆ Input data : key index + padded ciphered data, prefixed by a byte set to the padding length..
 - ◆ Output data : unpadded data.
- **3DES CBC signature** (*SIGN*)
 - ◆ Input data : key index + data to be signed.
 - ◆ The data is displayed to the user, who must verify CHV1.
 - ◆ Output data : signature (MAC4).
- 3DES operations are performed using two 8-byte keys.
- Key storage was out of scope but is now covered.

Universal 3DES Signature (*USGN*)

- This plug-in is similar to the Standard Security *SIGN* plug-in.
- 3DES CBC signature :
 - ◆ Input data :
 - *key index*
 - *text to be signed (including variables encoded in UCS2)*
 - **text encoding (GSM / UCS2)**
 - ◆ Output data : signature (MAC4).
 - ◆ The text is displayed to the user, who must verify CHV1.
- The 3DES signature is performed using two 8-byte keys.
- Key storage is out of scope.

PKI plug-ins

- WIB 1.3 supports several Public Key Infrastructure (PKI) plug-ins.
- Decryption : **Asymmetric Decryption** (*AD*)
- Digital signature (text) : **PKCS#1** (*P1*)
- Digital signature with time stamp (text) : **PKCS#7** (*P7*)
- Digital signature and non-repudiation (data) : **Fingerprint** (*FP*)
- **Public Key Retrieval** (*PKR*)
- The **S3S** plug-in (proprietary RSA signature) is dead.

PKI plug-ins personalization

- User dialogs are stored in transparent files, which are defined by SmartTrust.
- Key pairs are stored in proprietary transparent files.
 - ◆ AD : one key pair for decryption, one key pair for unwrapping.
 - ◆ FP : one key pair for non-repudiation, one key pair for authentication.
 - ◆ P1 : one key pair for signature.
 - ◆ P7 : one key pair for signature.
- The WIM files (PKCS#15) could also have been used.
- Access to each private key is protected by a PIN (PIN Management plug-in).

PIN Management (*PM*)

- All keys used by the crypto plug-ins are protected by a PIN. A user may forget its value or block it.
- *PM* provides administrative features for all these PINs.
- Change PIN : user operation
 - ◆ Private Key ID (Private key : Non-repudiation / Signature / Decryption / Unwrap).
 - ◆ Or: Symmetric Key ID.
- Reset PIN : server operation
 - ◆ Key ID.
 - ◆ New PIN value, ciphered and MAC'ed (3DES CBC (2 keys) + SHA1 or ISO9797_ALG3).

Asymmetric Decryption (AD)

- This plug-in deciphers a PKCS#1 ciphertext :
 - ◆ Input data : Ciphertext + key number.
 - ◆ Output data : Cleartext, with padding.
- 1. Display at most 16 bytes of the ciphertext (**optional**).
- 2. Ask the user to verify the PIN protecting the private key.
- 3. Decipher the ciphertext.
- 4. Return the cleartext to the browser.

Decrypt: 1234
AB12 D3661 1F01
2354 5CE1 1010
B210
Cancel Ok

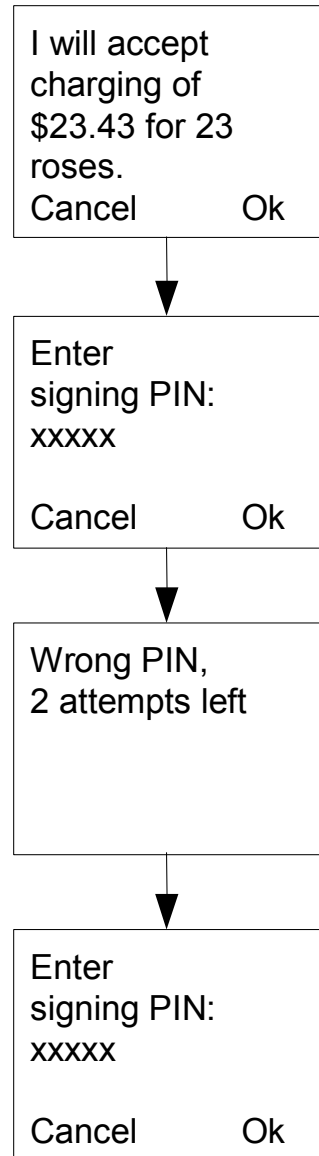
Enter
decryption PIN:
xxxxx
Cancel Ok

Wrong PIN,
2 attempts left

Enter
decryption PIN:
xxxxx
Cancel Ok

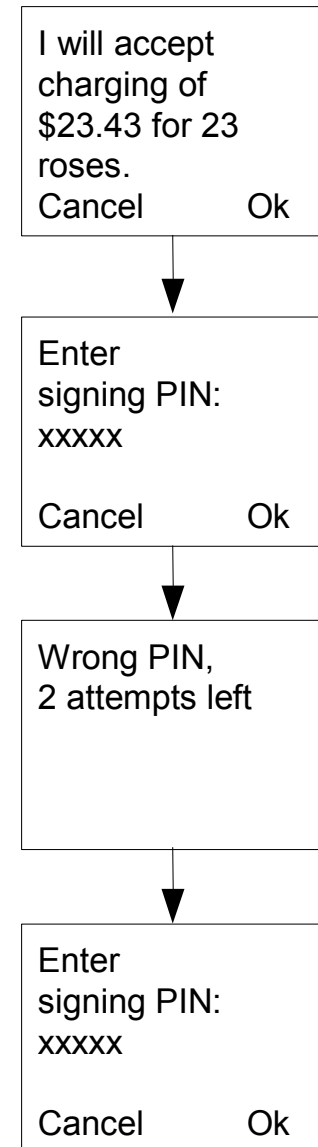
PKCS#1 (P1)

- This plug-in performs an application-level digital signature, which is output-compliant with PKCS#1 (see Appendix B).
 - It may be used to sign a small amount of **displayable text** : “What-you-see-is-what-you-sign”
1. Display to the text to be signed (GSM / UCS2)
 2. Ask the user to verify the PIN protecting the private key.
 3. Sign the text.
 4. Return the signature to the browser.



PKCS#7 (P7)

- This plug-in performs an application-level digital signature, which is output-compliant with PKCS#7 (see Appendix B).
 - ◆ It may be used to sign small amounts of **displayable text**, i.e. text entered on the mobile phone or received in a SMS, which also need to be **time stamped**.
 - ◆ It is mainly used for **non-repudiation** (similar to the signText WAP operation).
1. Display the text to be signed.
 2. Ask the user to verify the PIN protecting the private key.
 3. Sign the text.
 4. Return the signature to the browser.



Fingerprint (FP)

- This plug-in performs an application-level digital signature, which is output compliant with PKCS#1.
 - It may be used to sign :
 - ◆ “large” amounts of data, i.e. an e-mail message, a word-processing document.
 - ◆ data that is not displayable on a mobile phone, i.e. random data used during the set-up phase of a VPN (Virtual Private Network).
 - ◆ It is mainly used for authentication.
1. Display at most 16 bytes of the data to be signed (optional)
 2. Ask the user to verify the PIN protecting the private key.
 3. Sign the text.
 4. Return the signature to the browser.

Authorize: 1234
AB12 D3661 1F01
2354 5CE1 1010
B210
Cancel Ok

Enter
signing PIN:
xxxxx
Cancel Ok

Wrong PIN,
2 attempts left

Enter
signing PIN:
xxxxx
Cancel Ok

Public Key Retrieval (*PKR*)

- This is a **Vodafone** plug-in (Vodafone Generic 2 specification).
- It allows the WIG to **read the Public Key** associated to a given Private Key (Non-repudiation / Signature / Decryption / Unwrap).
- The plug-in also provides APDU commands to **read Public Keys** and to **generate key pairs**. This may be used a Point Of Sale.
- The output format is PKCS#1 compliant, i.e. the plug-in returns a DER-encoded value of the *RSAPublicKey* ASN.1 type :

```
RSAPublicKey ::= SEQUENCE {  
    modulus INTEGER, -- n  
    publicExponent INTEGER -- e  
}
```

Call Control

- 'Call Control' is a procedure defined in [51.014]. It is provisioned using the 'Call Control' activation bit in the SIM Service Table.
- When this service is activated, all **dialed digit strings**, **Supplementary Service** (SS) control strings and **Unstructured Supplementary Service Data** (USSD) strings are first passed to the SIM before the ME sets up the call/SS/USSD.
- The SIM also receives **location information**, **network code**, etc.
- The SIM may **allow**, **bar** or **modify** the call/SS/USSD.
- The SIM may **replace** a call/SS/USSD by another call/SS/USSD.

Call Control Script

- This feature is implemented in a **SIM Toolkit applet**, which interprets a **Call Control script** stored in EF_CALL_CONTROL_SCRIPT. The script is updateable by OTA.
- The script and the file are defined by a **Vodafone** specification.
- The Call Control applet is not a WIB plug-in : it doesn't receive WIB scripts.
- It receives call details from the ME in a ENVELOPE (CALL CONTROL) command.
- Once the applet has processed the call, it replies to the ME.
- **Optionally**, the applet stores the dialed number in EF_CALL_CONTROL and triggers the WIB using the CALL CONTROL event. The WIB may then read from EF_CALL_CONTROL and display a message to the user.

Call Control Script File

- The Call Control script file contains a list of **Case Objects**, a **Default Action** and an **Error Action**.
- When the applet is triggered, it applies each Case object to the call details.
 - A Case Object contains :
 - A **Test Object** : call details, with wildcards.
 - An **Action Object** : allow / deny / modify the call details.
 - If the test described in the Test Object is true, the action described in the Action Object is performed (possibly sending data to the WIB).
 - if not, move to the next Test Object.
- If none of the Tests Objects apply to the call details, the Default Action is performed.
- If anything goes wrong (and it will !), the Error Action is performed.

Call Control example (1/2)

Here is a simple Case Object, which reroutes any phone number starting with 014138 to to 0141381700.

EF_CALL_CONTROL_SCRIPT:

```
70 1a // Case Object

71 06 // Test Object
    06 04 // Address TLV (cf. 102.223)
    81 10 14 83 // Does address start with 014138 ?

72 10 // Action Object (Don't trigger the WIB)
    30 30 30 30 30 32 // Event label (unused)
    02 08 // Allowed with modifications
    06 06 // Route call to an address
    81 10 14 83 71 00 // Dial 0141381700
```

Call Control example (2/2)

Command :

```
a0 c2 00 00 17 // ENVELOPE command
    d4 15 // CALL CONTROL event
    82 02 82 81 // ME -> SIM
    06 06 81 10 14 83 71 69 // The user dialed 0141381796
    93 07 00 f1 10 00 01 00 01 // Location Information
    (9F xx)
```

Response :

```
a0 c0 00 00 W(2:2) // GET RESPONSE command
    [02 08 // Allowed with modifications
      06 06 // Route call to an address
      81 10 14 83 71 00] // Dial 0141381700
    (90 00)
```

References

- USAT Interpreter – <http://www.3GPP.org/>
- SIM Alliance – <http://www.SIMAlliance.org>
- SmartTrust – <http://www.SmartTrust.com>
- PKCS – <http://www.RSALabs.com>
- WAP – <http://www.OpenMobileAlliance.com/>
- World Wide Web Consortium – <http://www.w3.org/>

Appendix A : WML

WML : user input

- The **input** element defines an input field where the user may enter information.
- A \$ character followed by (VARIABLENAME) indicates a variable reference.
- This example will prompt the user to enter at most 20 digits :

```
<input title="Please enter your phone number"  
  type="text" name="PHONE" format="*N" maxlength="20"/>
```

WML : variables (1/2)

- The **setvar** element sets the value of a variable. Variable references used in the value of an attribute are replaced on a byte-per-byte basis in the WIB.
- A \$ character followed by (VARIABLENAME) indicates a variable substitution.
- This example will display the text "Item: CD Price: \$10" :

```
<card>
```

```
  <p>
```

```
    <setvar name="ITEM" value="CD"/>
```

```
    <setvar name="PRICE" value="10"/>
```

```
    Item: $(ITEM) Price: $$$ (PRICE)
```

```
  </p>
```

```
</card>
```

WML : variables (2/2)

- The WIB must support up to 252 variables per script.
 - ◆ Variable 0x00 is not allowed.
 - ◆ Variables 0x80, 0x81 and 0x82 are reserved for WIB use.
- Variables may be GSM or UCS2 encoded.
- Nested variables are not supported.
 - ◆ SmartTrust WML supports nested variables.
- The WIB supports INSANE variable management features:
Substring, Add/Subtract, Convert (hex-BCD-decimal), Group/
Ungroup, Swap Nibbles, GSM-UCS2 conversion.
- The SmartTrust specification recommends allocating 1 Ko of
EEPROM for variable storage.

WML : user selection

- The **select** element defines and displays a set of optional list items from which the user can select an item.
- An **option** element is required for each item in the list. The name of the menu, normally displayed by the mobile station, is specified by the **title** attribute.
- This example will prompt the user to select one item and jumps to the corresponding destination :

```
<select title="Please choose service" name="SELECTION">  
<option value="Bank" onpick="#CARD2">Banking</option>  
<option value="Gamble" onpick="#CARD3">Gambling</option>  
<option value="blah" onpick="http://server/  
home.wml">[Home]</option>  
</select>
```


WML : user navigation (1/2)

- The **anchor** (or **a**) element binds a text section to a **go** element. A sequence of anchors is presented to the user as a list of items, from which the user can select one.
- When the user has selected one item, the corresponding **go** is executed.
- This example will prompt the user to select one item and jumps to the corresponding destination :

Choose a service.

```
<anchor><go href="#CARD2"/>Banking</anchor>
```

```
<anchor><go href="#CARD3"/>Gambling</anchor>
```

```
<anchor><go href="http://server/home.wml"/>[Home]</anchor>
```

WML : user navigation (2/2)

- The **do** element is a general mechanism for the user to act upon the current card.
- The **go** element declares a jump to :
 - ◆ a card in the deck,
 - ◆ A URL : this is equivalent to a GET/POST HTTP request.
- The **go** element may also be used to perform WIB or WIG specific commands.
- This example will prompt the user for a string, then passes the string to a web server :

```
<input type="text" title="Enter name" />
```

```
<do type="accept">
```

```
  <go method="post" href="http://server/page.jsp?f=$ (NAME) " />
```

```
</do>
```

WML : `<go href>` element

- The **go href** element is used to perform a call to :
 - ◆ A SIM Toolkit command,
 - ◆ A WIB specific command,
 - ◆ A WIB plug-in,
- The syntax is similar to a call to a WML Script subroutine.
- The commands have different function names and different numbers of arguments.
- The arguments are used for both input and output data.

```
<do type="vnd.smarttrust.extended">
```

```
<go href="http://server#functionname('arg1','arg2',...)" />
```

```
</do>
```

Appendix B : PKCS

PKCS#1 : signature operation (1/2)

1. **Hash the text** to be signed (SHA1, 20-byte output).
2. **Encode the hash** in a *DigestInfo* ASN.1 type :

```
30 21                                -- SEQUENCE (DigestInfo)
  30 09                                -- SEQUENCE (AlgorithmIdentifier)
    06 05 2b 0e 03 02 1a              -- digestAlgorithm = sha-1
    05 00                                -- parameters = NULL
  04 14                                -- OCTET STRING (digest)
    xx xx xx xx                      -- digest value
    xx xx xx xx
    xx xx xx xx
    xx xx xx xx
    xx xx xx xx
```

PKCS#1 : signature operation (2/2)

3. The message to sign must have the same length as the key modulus :
generate *PS*, a padding sequence of random non-zero bytes.

4. Build the message to sign :

$$M = 01 \parallel PS \parallel 00 \parallel \text{DigestInfo}$$

5. Sign *M* with the private key (RSA modular exponentiation).

6. Output the result :

$$R = 01 \parallel \text{Modulus length} \parallel M$$

PKCS#7 : signature operation (1/2)

1. Hash the data to be signed (SHA1, 20-byte output).
2. Encode the hash in the following template :

```
31 57
 30 16
    06 09 2a 86 48 86 f7 0d 01 09 03 -- contentType
    06 09 2a 86 48 86 f7 0d 01 07 01 -- data
 30 1a
    06 09 2a 86 48 86 f7 0d 01 09 05 -- signingTime
    17 0d xx xx xx xx xx xx xx xx xx xx xx xx xx xx -- UTCTime
 30 21
    06 09 2a 86 48 86 f7 0d 01 09 04 -- messageDigest
    04 14 xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx
    -- SHA-1 digest
```

3. Sign the template according to PKCS#1 (cf. PKCS#1 plug-in).
4. Output the result (with optional data).

PKCS#7 : signature operation (2/2)

■ Input data

- ◆ Text to be signed (1 to 160 bytes).
- ◆ Text encoding : GSM or UCS2.
- ◆ Option flags.

■ Output data

- ◆ Signature (i.e. DER-encoded value of the *DigestInfo* ASN.1 type).
- ◆ Text to be signed (optional).
- ◆ Hash of the public key (optional).
- ◆ URL of the public key certificate (optional).
- ◆ ICCID of the SIM card (optional).
- ◆ Message digest of the text to be signed (optional).

Fingerprint : signature operation (1/2)

- The signature operation is identical to the one performed by the PKCS#1 plug-in.
- Depending on the input flags, the output may include optional data :

```
R = 01 || Modulus length || M
      || Total length || 80 || ICCID
                                || 01 || Public key hash
                                || URL
```

Fingerprint : signature operation (2/2)

■ Input data

- ◆ Data to be signed (DER-encoded value of the *DigestInfo* ASN.1 type).
- ◆ Option flags.

■ Output data

- ◆ Signature.
- ◆ Hash of the public key matching the private key (optional).
- ◆ URL of the public key certificate (optional).
- ◆ ICCID of the SIM card (optional).