1 Introduction

SEMS is a new protocol for encrypting and signing short messages in such a way that the resulting output is optimized for transmission using the GSM Short Message Service. SEMS is a free protocol: everyone can implement it, and no components used for SEMS are patented, licensed or otherwise encumbered.

This document describes everything needed to understand and implement SEMS.

2 The SEMS protocol

2.1 About SEMS operations

All SEMS users generate a public/private key pair. There are currently two types of keys: a 1112 we sometimes call a “medium security key”, or a 2208 bit RSA key which we refer to as a “high security key”. Messages encrypted or signed using the first key size precisely fit one SMS message, messages encrypted or signed with the second key size fit 2 SMS messages.

There exist three basic types of message objects to be sent using SEMS. They are

- text messages
- keys and
- key revocations

Objects sent via SEMS can be encrypted or sent in the clear. The objects can also be signed, or they can travel unsigned.

2.1.1 Signatures

A signature can be added in two ways. If the object is short enough, an included signature can be used. This means the object and the signature can travel the same SMS (or the same two SMSs in the case of a high security signature). Included signatures save on the number of SMSs sent, but are only possible under the following circumstances:

- The message must fit inside one transport object for the desired signature size
- The recipient must be able to tell who the sender is
- The sender must be certain that the recipient has her key

If one or both of the last two conditions are not met, the message is sent, but gets lost because the recipient SEMS client has no idea what to do with it. This can happen for instance because the SMS message is relayed in some way that loses the phone number of the sender, or because the recipient doesn’t have the sender’s key

To determine the message length for which an included signature is still possible, take the maximum payload size for the transport object that fits the signature type (989 bits for medium security, 2084 bits for high security). From this number, deduct 192 bits for header and signature information, leaving 797 or 1892 bits respectively for symbol size field (2 bits) and message.

2.2 Unencrypted

2.2.1 Without signature

To send unencrypted, the text message (5.2) or key (5.3) is wrapped in multiple multipart containers (5.6) if it is larger than the maximum payload size for an unencrypted transport object (4.1), which is 1069 bits. These containers are then packed in these unencrypted transport objects and sent.

2.2.2 With medium security included signature

- Form a message object with included signature. Set the signature type field to included, and insert a timestamp and the appropriate SHA-1 hash output in the signature information field.
- Insert this object in a medium security envelope, including the same 80 random bits that were also used in the SHA-1 hash of the previous step
- Perform a Shake and a secret key operation with the secret key of the sender on the envelope
- Insert the result in a medium security transport object (4.2.1), setting the content description to 10 to indicate that this is an unencrypted message with an included signature.
- Send the transport object to the recipient

2.2.3 With high security included signature

Unencrypted with a high security included signature is done in almost the same way as unencrypted with a medium security included signature. But in step 2 the message object is put inside a high security crypto envelope (4.3.2) and this envelope is encrypted and then inserted in the two high security transport objects: (4.3.1.1 & 4.3.1.2).

2.2.4 With detached signature

- To send any object with a detached signature, the signature is built first. For this, we form a detached signature object (5.5). If the object to be signed is a text message, the SHA-1 in the signature object is made over “symbol size || message content || timestamp || 80 random bits”, if the object is a key, the SHA-1 is done over “fingerprint || timestamp || 80 random bits”.

- This object is inserted in a crypto envelope of the right size for the signing key (4.2.2 for medium security, (4.3.2 for high security). The 80 bits of random in the envelope are the same as what was used in the SHA-1 in the previous step

- A Shake and an RSA secret key operation with the key of the sender are performed on the envelope. See paragraph 3.2.2 for details on the secret key operation.

- The first 20 bits of the fingerprint of the signing key, and the first 20 bits of the resulting envelope of the previous step make up a signature description.

- Then the actual message (either text message or key) is composed, using the 40-bit signature description in the appropriate field in the header to denote that a signature is to be expected for this message. Otherwise the message travels as it would if it were unsigned, that is: no special operations are performed in composing the message.

2.3 Medium security encrypted messages

2.3.1 Without signature

- Form a message object
- Insert this object in a medium security envelope
- Perform a public key operation with the public key of the recipient on the envelope
- Insert the result in a medium security transport object (4.2.1)
- Send the transport object to the recipient

2.3.2 With medium security included signature

- Form a message object with included signature. Set the signature type field to included, and insert a timestamp and the appropriate SHA-1 hash output in the signature information field.
- Insert this object in a medium security envelope, including the same 80 random bits that were also used in the SHA-1 hash of the previous step
- Perform a secret key operation with the secret key of the sender on the envelope
- Reset the most significant bit of the envelope
- Perform a public key operation with the public key of the recipient on the envelope
- Insert the result in a medium security transport object (4.2.1)
- Send the transport object to the recipient

2.3.3 With high security included signature

A high security envelope would need to be packed to travel through a medium security encrypted channel. Although this would be possible using an encapsulation packet inside multipart containers, it would take 3 SMS messages. Since detached signatures are superior to included signatures, and doing this with a detached signature would also take 3 SMS messages, we use that method instead.

2.3.4 With detached signature

- To send any object with a detached signature, the signature is built first. For this, we form a detached signature object (5.5). If the object to be signed is a text message, the SHA-1 in the signature object is made over “symbol size || message content || timestamp || 80 random bits”, if the object is a key, the SHA-1 is done over “fingerprint || timestamp || 80 random bits”.

- This object is inserted in a crypto envelope of the right size for the signing key (4.2.2 for medium security, (4.3.2 for high security). The 80 bits of random in the envelope are the same as what was used in the SHA-1 in the previous step

- The key and IV for symmetric crypto are determined using the method described in 3.3.1

- An RSA secret key operation with the key of the sender is performed on the envelope. See paragraph 3.2.2 for details on the secret key operation.

- A symmetric crypto operation (3.3) is performed on the envelope.

- The first 20 bits of the fingerprint of the signing key, and the first 20 bits of the resulting envelope of the previous step make up a signature description.

Then the actual message (either text message or key) is composed, using the 40 bit signature description in the appropriate field in the header to denote that a signature is to be expected for this message. Otherwise the message travels as it would if it were unsigned, that is: no special operations are performed in composing the message.

2.4 High security encrypted messages

2.4.1 Without signature

- Form a message object
- Insert this object in a high security envelope (4.3.2)
- Perform a public key operation with the public key of the recipient on the envelope
- Insert the result in high security transport objects (4.3.1.1 and 4.3.1.2)
- Send the transport object to the recipient

2.4.2 With medium security included signature

- Form a message object with included signature. Set the signature type field to included, and insert a timestamp and the appropriate SHA-1 hash output in the signature information field.
- Insert this object in a medium security envelope, including the same 80 random bits that were also used in the SHA-1 hash of the previous step
- Perform a secret key operation with the secret key of the sender on the envelope
- Encapsulate the result in an encapsulation object (5.7)
- Pack the encapsulation object in a high security crypto envelope (4.3.2)
- Perform a public key operation with the public key of the recipient on the envelope
- Insert the result in the two high security transport objects (4.3.1.1 and 4.3.1.2)
- Send the transport object to the recipient

2.4.3 With high security included signature

- Form a message object with included signature. Set the signature type field to included, and insert a timestamp and the appropriate SHA-1 hash output in the signature information field.

- Insert this object in a medium security envelope, including the same 80 random bits that were also used in the SHA-1 hash of the previous step
- Perform a secret key operation with the secret key of the sender on the envelope
- Reset the most significant bit of the envelope
- Perform a public key operation with the public key of the recipient on the envelope
- Insert the result in two high security transport objects (4.3.1.1 & 4.3.1.2)
- Send the transport object to the recipient

2.4.4 With detached signature

Same as 2.3.4

2.5 Allowed operations

Not every payload data type can be used with each mode of operation. For instance, a key revocation cannot be sent without a signature of the key to be revoked. This means an unencrypted packet or unsigned encrypted packet with a key revocation is not to be sent, and has to be rejected when received.
Table 1 shows all the modes of operation, and the transport object (1.x) and data objects (5.x) that can be used for this mode.

Table 1

Signature / **Encryption**	No signature	Medium security signature included	High security signature included	Content claiming detached signature	Medium security detached signature	High security detached signature
Unencrypted	4.1 5.2, 5.3, 5.6	4.2 5.2, 5.3, 5.4 [e]	4.3 5.2, 5.3, 5.4	4.1 5.2, 5.3, 5.4, 5.6	4.2 5.5	4.3 5.5
Encrypted to Medium Security key	4.2 5.1, 5.2, 5.3, 5.6	4.2 5.2, 5.3, 5.4 [ae]	Use detached signature [b]	4.2 5.2, 5.3, 5.4, 5.6	4.2 5.5 [d]	4.3 5.5 [d]
Encrypted to High Security key	4.3 5.1, 5.2, 5.3, 5.6	4.3 5.2, 5.3 or 5.4 inside 5.7 [ce]	4.3 5.2, 5.3, 5.4 [a]	4.3 5.2, 5.3, 5.4, 5.6	4.2 5.5 [d]	4.3 5.5 [d]

a) Two RSA operations are performed, first with the secret key of the sender, then with the public key of the recipient. The MSB of the output is reset to zero after the first operation.
b) This operation would involve packing a long signature in three short packets using encapsulation. Since detached signatures are superior and this operation would also take three SMSs using a detached signature, we use that instead.
c) An encapsulation packet (5.7) is used to transmit the shorter signature output over 2 SMS messages.
d) Signature is not encrypted with public key, but with the AES symmetric algorithm. The key and IV are determined from the (first packet of the) message.
e) Although 5.3 (key) is listed, the current key format never fits inside one medium security included signature packet. This may change with a new key format, and thus 5.3 is listed here.

2.6 Examples

2.6.1 An unencrypted text message with a medium security included signature

A medium security included signature on a unencrypted text message would made thus:

- Form text message object (5.2) as follows:

field	**content**	**comment**
payload data type	0001	Text message
signature type	10	included
signature info	SHA-1 over "symbol size || message content || timestamp || 80 random bits"	
	26-bit timestamp	
symbol size	10	7 bit symbols
message	"Hello World" in 7 bit characters	

- Form a medium security crypto envelope (4.2.2) as follows:

field	**content**	**comment**
set to zero	0	
length	10 bits, length of object made above	
payload	object made above	pad with zeroes until 989 bits
random	80 bits	Use the same bits that were used for the signature info hash above.

- Perform the necessary operations on the crypto envelope. In this particular case, a Shake operation and a private key operation with the secret key of the sender are performed.

- Form a medium security transport object (4.2.1) as follows:

field	content	comment
SEMS version	000	SEMS v1
packet size	0	1 SMS
content desc.	10	unencrypted with medium security included signature
reserved	00	
crypto envelope	crypto envelope	The resulting crypto envelope formed in the previous step.

- Send the transport object to the recipient

3 Basic functions

These basic mathematical functions form the building blocks upon which the rest of the SEMS protocol is built. With the exception of key generation, the basic functions of SEMS are performed on chunks of data called 'crypto envelopes'. A crypto envelope contains some header information, a SEMS data object and 80 bits of random padding. Crypto envelopes come in two sizes: either 1112 bits (see 4.2.2) or 2208 bits (see 4.3.2). After performing the necessary basic functions on the crypto envelope, the envelope is packed in either one or two transport objects and sent.

3.1 Shake

Shake (which stands for **S**ome **H**ashes **A**gainst **K**nown **E**xcrements) is a fully invertible function that involves lots of SHA-1 hashes and serves no other purpose than to fully randomize the input bits to the RSA function in a mathematically complex way. Shake makes sure that a one bit change in the input affects all output bits. This is necessary to foil some otherwise dangerous attacks against systems using RSA with a relatively large message and a small public exponent.

Shake is performed on all but the first bit of the crypto envelope. This bit is not passed to Shake and has to remain zero in order to make sure that the resulting number can never be too large for the RSA function to handle.

3.1.1 Shake pseudocode

To perform Shake, the input is split into 160-bit chunks, leaving a smaller chunk at the end. The chunks are numbered M0 through Mr, where r is one less than the total number of chunks, and Mr is likely to be smaller than 160 bits.

```
M0' = M0 xor SHA-1(0x00 || M1 || … || Mr)

FOR j = 1 TO r

    Mj' = Mj xor SHA-1( [j mod 256] || M0')

NEXT

M0'' = M0' xor SHA-1( [r+1 mod 256] || M1' || M2' || M3'[bits 0
through 111])

Result_of_Shake = M0'' || M1' || … || Mr'
```

Notes: || is used to denote concatenation of the bit strings. The 0x00 in the first line, the j that is input to the SHA-1 inside the FOR-NEXT loop and the r+1 that is input in the last SHA-1 are all 8-bit values. When the loop comes to r, Mr' is taken to mean only the first part of the SHA-1 output, up to the length of Mr.

3.1.2 Inverting Shake

When Shake needs to be undone, the input of the inverting function is the result of a Shake operation, i.e.: M0'' || M1' || … || Mr'. To undo this, perform:

```
M0'= M0'' xor SHA-1( [r+1 mod 256] || M1' || M2' || M3'[bits 0
through 111])

FOR j = 1 TO r

        Mj = Mj' xor SHA-1( [j mod 256] || M0')

NEXT

M0 = M0' xor SHA-1(0x00 || M1 || … || Mr)

Output_of_inversion = M0 || … || Mr
```

3.1.3 Shake scientific background

The idea of Shake is the randomization of the input of the RSA function. After Shake every bit depends on every other bit in a complex way. We use a SHA-1 based pseudo random function (SHA-1(j || .) with a byte counter j). Shake is only a slight modification away from the BEAST algorithm constructed by Stefan Lucks in 1998 [1].
To make sure we have a strong theoretical basis we use an unbalanced Feistel construction resp. unbalanced Luby/Rackoff Ciphers [2]. This construction element is widely used and there are a lot of mathematical security proves that show a close relation between the security of the roundfunction and the whole output. Since we use the well-trusted SHA-1 hashfunction we have a huge margin of security.
Compared to simple mixing constructions we add a fast 3rd round. This is done because the shortcut theorem by Stefan Lucks [3] showed that a 160-bit input in the 3rd round is sufficient for the security proves. We use 432 bits as input for the last SHA-1 because 512-(64-1)-8=439 is the biggest possible input for which SHA-1 does its internal timeconsuming compression function only once [4] [5]. We use 432 instead of 439 because it is byte aligned.

---

1 Lucks, S., "BEAST: A fast block cipher for arbitrary blocksize", IFIP'96, Conference on Communication and Multimedia Security, Chapman & Hall, 1996, pp. 144--153.

2 Luby, M., Rackoff, C., "How to construct pseudorandom permutations from pseudo random functions", SIAM J. Computing, Vol 17, No. 2, 1988, pp. 239--255.

3 Lucks, S., "Faster Luby-Rackoff ciphers", Fast Software Encryption, Springer LNCS 1039, 1996.

4 Weis, R., "Cryptographic Protocols and Algorithms for Distributed Multimedia Systems", PhD thesis, Shaker Verlag, Aachen, Maastricht, 2000.

5 Weis, R., Lucks, S., "Fast Multimedia Encryption in JAVA Using Unbalanced Luby/Rackoff Ciphers", 4th European Conference on Multimedia Applications, Services and Techniques, ECMAST'99, Springer LNCS 1629, Madrid, 26-28 MAY 1999.

3.2 Public/private key operations (RSA)

All RSA functions are performed treating the contents of the crypto envelope as an integer of which the MSB is at the beginning of the envelope (and is always set to zero). The public key operations of encrypting and signature verification differ at a higher level of the SEMS protocol, but are exactly the same in terms of what is done to the number in the crypto envelope. The same goes for the secret key operations of signing and decrypting.

3.2.1 Public key operations

The encryption envelope after a public key operation holds the output c of the function:

$$c = m^e \bmod n$$

where m is the original contents of the crypto envelope, n is the public modulus and e is the public exponent, which is fixed to the number 3 in the case of SEMS type 1 keys. In the rest of this document, any reference to "public key" only refers to this public modulus.

3.2.2 Secret key operations

Secret key operations are a little harder, because we use the Chinese reminder theorem to speed up calculations. The crypto envelope after a secret key operation contains the output m of the last function.

$$m_1 = c^{\,d \bmod (p-1)} \bmod p$$

$$m_2 = c^{\,d \bmod (q-1)} \bmod q$$

$$h = u * (m_2 - m_1) \bmod q$$

$$m = m_1 + h * p \bmod n$$

where c is the original contents of the crypto envelope, n is the public modulus and d,p,q and u are numbers we kept when we generated the key pair to allow for speeding up the secret key calculation using the Chinese reminder theorem.

3.3 Symmetric packet encryption (AES)

Symmetric encryption is done by performing AES in CBF mode on the entire crypto envelope, using a 256-bit key and a 128-bit initial vector. Currently we only use symmetric encryption when we send an encrypted, detached signature packet. Further use of symmetric encryption is foreseen in the upcoming remailer definition.

3.3.1 Obtaining AES key and IV for detached signature encryption

Currently symmetric encryption is only used for encryption of detached signatures, and the key and IV are obtained from the first packet of the message that is signed. The sender takes the

contents of the crypto envelope of the message to be signed, after adding the random bits, but before performing any operations such as Shake or RSA on the envelope.

K_1 = SHA-1 (0x00 || envelope) (0x00 is 8 zero bits)
K_2 = SHA-1 (0x0000 || envelope) (0x0000 is 16 zero bits)

X = K_1 || K_2{first 96 bits)

256 bit AES key = K_1 || K_1 XOR K_2
Initial Vector = K_2{last 128 bits }

3.4 Key generation

During key generation, two prime numbers are generated. The smaller of the two we name *p*, the larger we call *q*. The length of these prime numbers is exactly half the desired keylength. The two most significant bits must be set, as well as the lowest bit (prime numbers are odd by nature). The other bits are filled with random and a test is done to see whether *p*-1 is divisible by three. If not, a test is done to see if the number is prime. If the number is not prime, the random is changed (adding 1 would suffice), and the tests are done again.

As soon as these two primes are found, the following math is performed:

$n = p * q$

$phi = (p-1) * (q-1)$

$e = 3$

d = Multiplicative inverse of e mod phi

u = Multiplicative inverse of p mod q

Multiplicative inverses are found using Euclid's algorithm. *n* is the public modulus and is published as the SEMS public key. *d*, *p*, *q* and *u* are kept locally as SEMS secret key information.

4 Transport objects

4.1 Unencrypted

bits	description	comment
3	SEMS version	000 = SEMS v1
1	SEMS envelope size	0 = 1112 bits
1	SEMS content desc.	0 = Unencrypted
3	reserved	Set to 000
11	Length of data in bits	
1069	Payload, pad with zeroes	
32	CRC-32 checksum	

4.2 Medium security transport

4.2.1 Medium security packet format

bits	description	comment
3	SEMS version	000 = SEMS v1
1	SEMS envelope size	0 = 1112 bits
2	SEMS content desc.	10 = Unencrypted with medium security included signature 11 = Other medium security traffic
2	reserved	Set to 00
1112	Medium security crypto envelope (4.2.2)	

4.2.2 Medium security crypto envelope

bits	description	comment
1	Set to zero	Number has to always be smaller than n
10	Length of data in bits	
989	Payload, pad with zeroes	
80	Random padding	
32	CRC-32	

4.3 High security transport

4.3.1 High security packet format

4.3.1.1 High security packet #1

bits	description	comment
3	SEMS version	000 = SEMS v1
1	SEMS envelope size	1 = 2208 bits
8	Packet ID	
1	Subpacket	0 = first packet
1	SEMS content desc.	0 = Unencrypted with high security incl. sig. 1 = Other high security traffic
2	reserved	Set to 00
1104	First half of high security crypto envelope (4.3.2)	

4.3.1.2 High security packet #2

bits	description	comment
3	SEMS version	000 = SEMS v1
1	SEMS envelope size	1 = 2208 bits
8	Packet ID	
1	Subpacket	1 = second packet
3	reserved	Set to 000
1104	Second half of high security crypto envelope (4.3.2)	

4.3.2 High security crypto envelope

bits	description	comment
1	Set to zero	Number has to always be smaller than n
11	Length of data in bits	
2084	Payload, pad with zeroes	
80	Random padding	
32	CRC-32	

5 Data objects

5.1 No Operation

bits	description	comment
4	Payload data type	0000 = No Operation

The receiving SEMS client shall ignore an incoming object of this type. No operation messages can be used to create ‘cover traffic’, to help foil traffic analysis. No operation messages cannot be signed, and they cannot be sent in the clear.

5.2 Text message

bits	description	comment
4	Payload data type	0001 = Text message
2	Signature type	00 = none 01 = detached 10 = included
0, 40 or 186	Signature information	See text
2	Symbol size	00 = 5 bits (modified Baudot) 01 = 6 bits (ASCII 0x20-0x5F, caps only) 10 = 7 bits (ASCII) 11 = 8 bits (binary, not displayed)
Variable	Message content	

5.2.1 Included signature information

In the case of an included signature, a 160 bit SHA-1 hash of “symbol size || message content || timestamp || random”, followed by the 26-bit timestamp is inserted as signature information.

5.2.2 Detached signature information

In the case of a detached signature, a 40-bit signature identifier is included. The first 20 bits of this identifier are the first 20 bits of the key fingerprint of the signing key. The second 20 bits are the first 20 bits of the signature packet crypto envelope in its encrypted form. This information enables the receiver to recognize this otherwise illegible block as the signature and to know who signed it.

5.3 Key

bits	description	comment
4	Payload data type	0010 = Key
2	Signature type	00 = none 01 = detached 10 = included 11 = both
5	Number of detached signatures	
Variable	Signature info	see text
1	Send me yours	1 = please send me your key
3	Key type	001 = RSA with public exponent 3
50	Telephone number	In international format, as integer
16	Expiration date	In days. 0 = never, 1 = August 1st, 2001
6	Length of Key Name field	In 7 bit characters
Variable	Key Name field	ASCII
1112 or 2208	Public Key	

If one user sends a key to another, the client could check whether there is a key stored for this other user. If this is not the case, the sending user is offered the option of including a request for the key of the recipient. If the user selects yes, the'Send me yours' bit in the key header will be set. If the recipient receives a key with this bit set, it offers the user the option of sending her key in return.

5.3.1 Key fingerprint

A SEMS key fingerprint is the output of a SHA-1 hash whose input is a concatenation of all fields starting at 'Key type'.

5.3.2 Included key signature

Although the key data type supports an included signature, currently only medium security keys signed by a high security key fit in one high security packet, and can thus have an included signature. If there is an included signature, the signature information field starts with a 160-bit SHA-1 hash over "fingerprint || timestamp || random", followed by the 26-bit timestamp.

5.3.3 Detached key signatures

If there are detached signatures, the signature information then continues with 40 bits per detached signature. These 40 bits describe the signature in the same way as with signatures on text messages (see 5.2.2).

5.4 Key revocation

bits	description	comment
4	Payload data type	0011 = Key revocation
2	Signature type	01 = detached 10 = included
40 or 186	Signature info	see text
160	Fingerprint of key that is being revoked	

The signature info holds 160 bits SHA-1 over “fingerprint || timestamp || random” followed by the timestamp in case of an included signature, or a 40 bit signature identifier in the case of a detached signature.
Key revocations have to be signed by the key to be revoked. If the signature is detached, the usual 40-bit signature identifier is used to help the recipient find the signature. Unsigned key revocations may not be sent, and are to be discarded without presentation to the user.

5.5 Detached signature

bits	description	Comment
4	Payload data type	0100 = Detached signature
160	SHA-1 hash of object	
26	time stamp	In minutes since jan 1st

The SHA-1 hash in the signature packet is calculated over "symbol size || message content || timestamp || random" or "key fingerprint || timestamp || random", where random is the 80 bits of random padding of the signature packet.

5.6 Multipart container

bits	Description	Comment
4	Payload data type	0101 = Multipart container
5	ID	
5	Number of this part	
5	Total number of parts	
Variable	Data	

Some text messages and keys to be sent are longer than the payload size for the packet used. In this case the data type is packed inside multiple multipart containers. The ID is a 5-bit number to distinguish this multipart transmission from other multipart transmissions between this sender

and recipient. The total number of parts cannot exceed 31. Multipart transmissions cannot have included signatures.

5.7 Encapsulation packet

bits	Description	Comment
4	Payload data type	0110 = Encapsulation packet
variable	Packet data	

Used to send a 1112 or 2208-bit crypto envelope when the transport encapsulation has a different block size. This packet is currently only used to encapsulate the output of a medium security included signature operation to allow it to travel encrypted to a high security key.

6 Appendices

6.1 Appendix A: Modified Baudot

Hex	Letters	Figures
00	.	*
01	E	3
02	,	LF
03	A	-
04	Space	Space
05	S	'
06	I	8
07	U	7
08	CR	CR
09	D	$
0A	R	4
0B	J	\
0C	N	,
0D	F	!
0E	C	:
0F	K	(
10	T	5
11	Z	"
12	L	)
13	W	2
14	H	#
15	Y	6
16	P	0
17	Q	1
18	O	9
19	B	?
1ª	G	&
1B	FIGS	%
1C	M	.
1D	X	/
1E	V	;
1F	@	LTRS

7 Index