Nethemba
all about security

# Public transport SMS ticket hacking

Ing. Pavol Lupták, CISSP, CEH
Lead Security Consultant

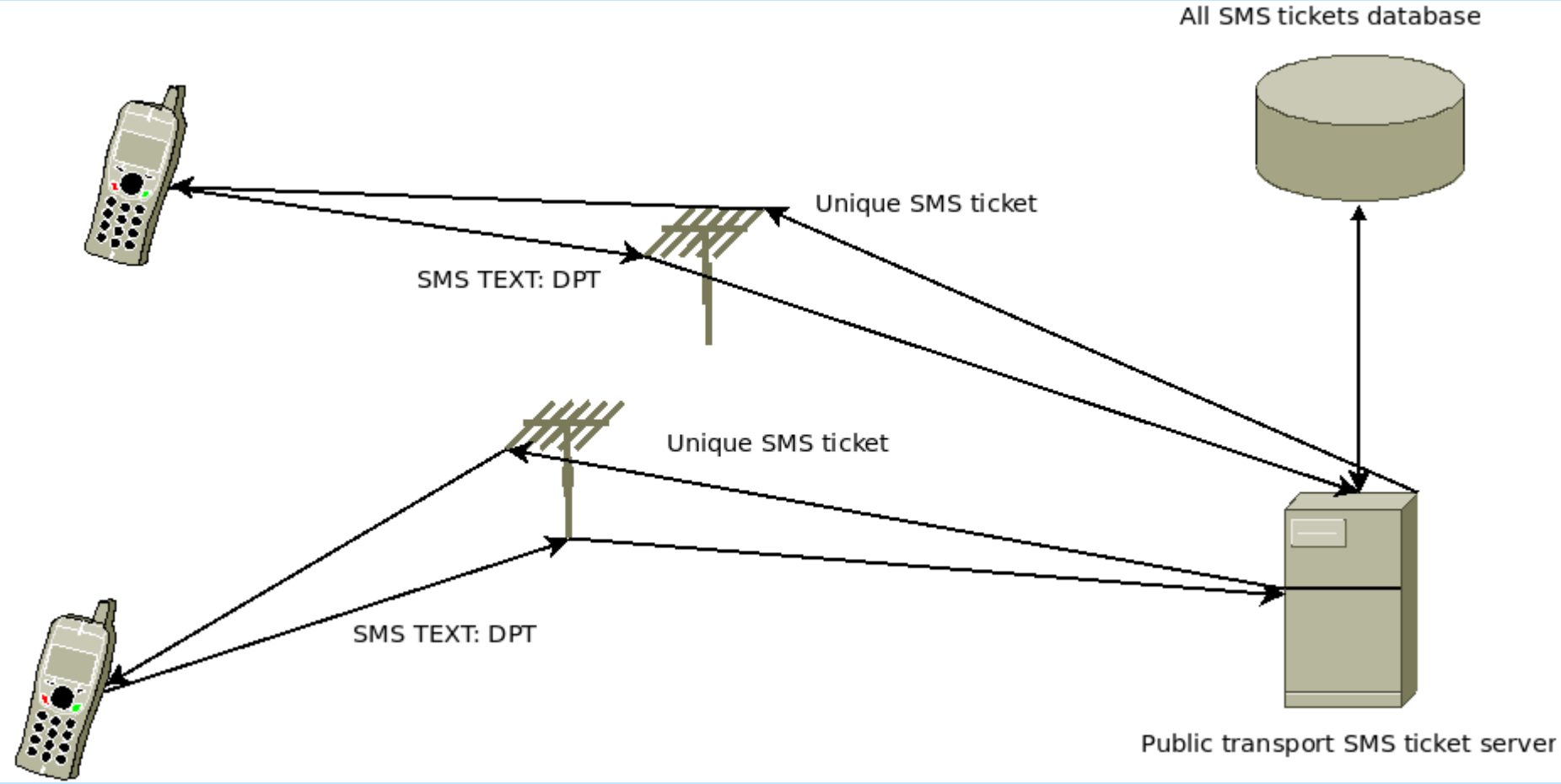
# Public transport SMS tickets - basics

- SMS request with a predefined body (e.g.“DPT”) is sent to a predefined number
- a unique SMS response (as a valid SMS ticket) is received in a few minutes:
- `Public Transport Company ABC, SMS ticket Price XYZ, Validity: from 28.10.07 13:20 to 28.10.07 14:50, code YrQPtMKs7 /52845`
- possibility to request for a duplicate if the SMS ticket is accidentally deleted or mobile phone is out of battery

# Who is still vulnerable?

- the biggest public transport companies in Czech Republic, Slovakia, Poland and Austria
- despite the fact that public transport companies have already been informed about this serious vulnerability, they ignore this fact and still use the vulnerable systems
- we have no information if there are some public transport SMS tickets that are not vulnerable to this kind of attacks

# SMS ticket architecture



# Looking for security problems

- SMS token (YrQPtMKs7 / 52845) seems to be sufficiently random and hardly predicted
- the inspector uses his PDA for online verification
- **BUT:** there is NO connection between user's identity and his SMS ticket's identity, therefore SMS ticket **can be SHARED by community**
- legislatively it is prohibited to generate and distribute copies of SMS tickets **but currently there is NO WAY to detect these duplicates**

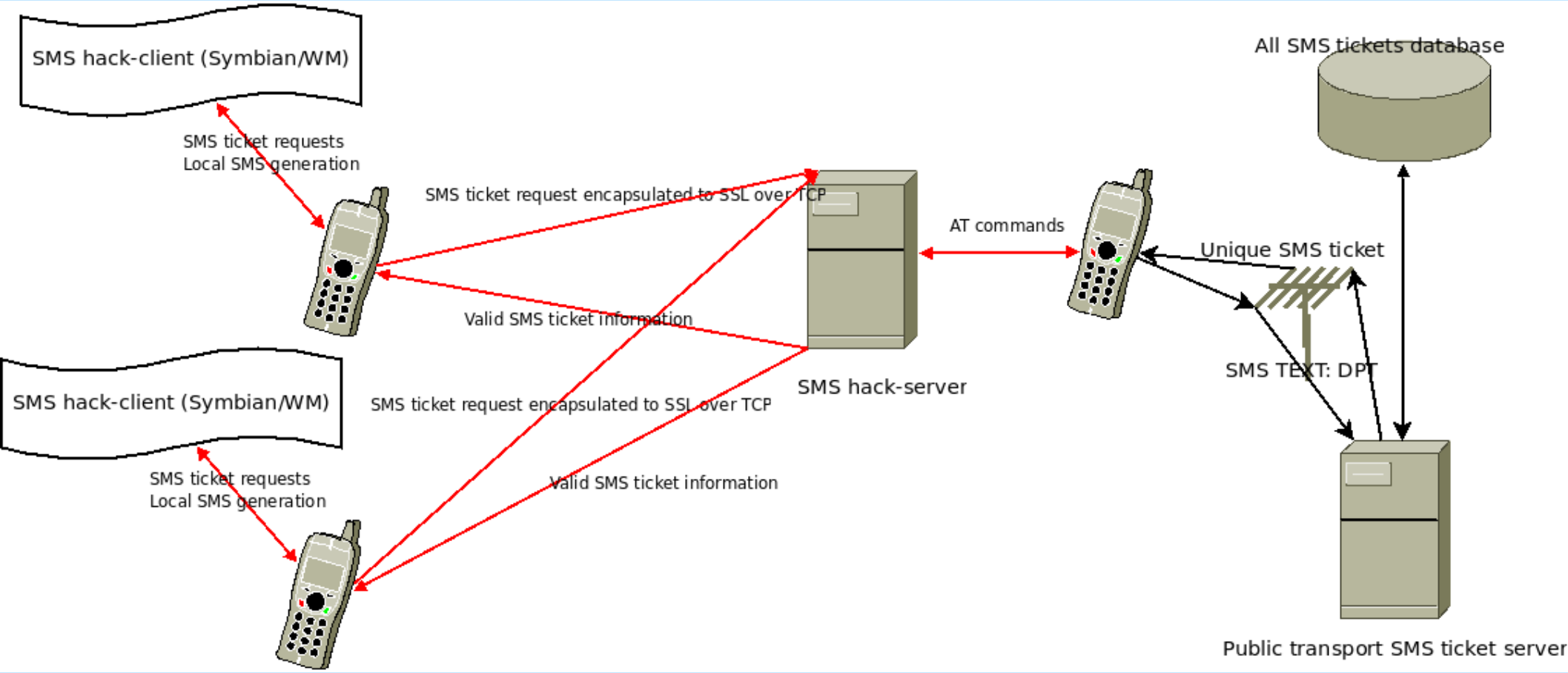
# Let's go to hack: SMS generation

- **an arbitrary SMS message can be locally generated:**
- every smartphone (Symbian, WM5, WM6) has capability to generate a local SMS with predefined sender, recipient, body and status flag (Not-Read-Yet)
- **practically verified:** Symbian S60v3 - we were able to generate a local SMS with a predefined recipient and additionally modify its sender and status-flag

# Let's go to hack: SMS distribution

- **SMS channel**
  - too expensive
  - impossibility to modify SMS headers (sender, recipient, status-flag)
- **TCP/IP**
  - very cheap (short SMS ticket requests/responses)
  - need of prepaid mobile data service
  - need of intelligent phone that is capable to generate local SMSes (any smartphone)

# Our hacked SMS ticket architecture



# SMS hack server

- responsible for sending SMS requests (“DPT”), receiving SMS responds (“SMS tickets”) through connected mobile-phone
- low-level AT commands over USB/serial/bluetooth line are used or high-level library (libgammu, libgnokii) functions are used
- simple one-thread application that listens on predefined TCP port and handles multiple SMS hack clients using poll()/select()

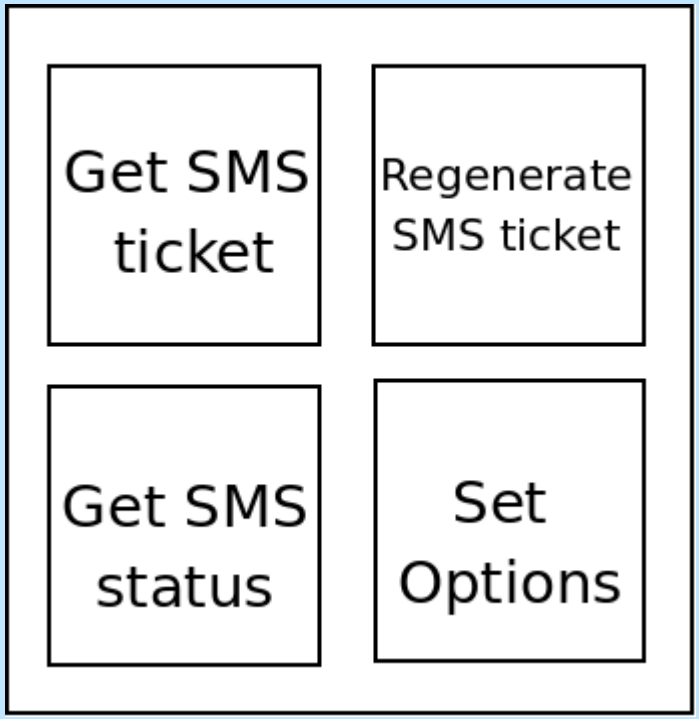
# SMS hack server II

- parses and extracts received SMS tickets:
  - ticket's validity (start time, expiration time)
  - unique ticket ID (that is used for inspection)
- extracts hack client user ID (gained from CN of the client certificate after SSL/TLS handshake)
- holds/updates information about which hack clients were asked for SMS tickets (if the given SMS ticket expires, information about given hack client is removed)

# SMS hack client

- always initializes TCP connection to the SMS hack server to predefined port (e.g. 80) (passes through multiple NATs/firewalls)
- sends specific requests to the SMS hack server (**Get_Token_Request, Regenerate_Token_Request, Get_Status_Request, Set_Option_Request, Send_GEO_Position**)
- receives valid SMS information from the SMS hack server that is used for generation of valid SMS tickets

# Hack client GUI



# Inefficient security solutions

- there are many inefficient ways how to detect our described attack, all of them can be circumvented by:
- **regeneration of already checked tickets** (in case the inspector looks for duplicate SMS tickets or uses sophisticated geographical correlation)
- **full MITM attack between the inspector and passenger mobile phone** (calls / SMS)
- **using of a private P2P mobile network** (blacklisting of the number responsible for sending valid SMS requests becomes impossible)

# Fix: Looking for duplicate SMS tickets

- inspector can look for duplicate SMS tickets (if two or more people use the same ticket, it is cleared that the shared ticket is used and distributed)
- **hacker's workaround:**
  - Every user must send the **Regenerate_Token_Request** after each inspector's check – a new valid SMS ticket is distributed to all clients by the central SMS hack server

# Fix: Looking for duplicate SMS tickets

- sometimes regeneration of SMS tickets can be too slow and it doesn't work when 10 passengers with the same ticket enter to the same bus with the inspector
- but there is a solution! “**passenger's geographical collision detection**”
- can be difficult to implement but it should work!

# Passenger's geographical collision detection

- SMS hack clients will send the passenger's current position to the central SMS hack server:
- **GPS coordinates** (if the given phone has GPS)
- **BTS names/signal levels** (that is used for evaluation of approximate position → Google Maps does it in this way)
- According to this data, the central SMS hack server will evaluate possible collisions and invalidate "collision" SMS tickets and ask for the new valid ones

# Fix: Sophisticated geographical correlation

- Public transport service can use sophisticated geographical correlation systems to reveal that one SMS ticket is used in a short time on multiple places that are too far from each other
- **hacker's workaround:**
  - Every user must send the **Regenerate_Token_Request** after inspector's check – a new valid SMS ticket is distributed to all clients by the central SMS hack server

# Fix: The inspector can make a call to the SMS ticket sender

- The inspector can make a call to the SMS ticket sender (because he knows his number)
- **hacker's workaround:**
  - Use GSM card / Asterisk VOIP call center on the central SMS hack server, if any call to this number is detected, Asterisk will make a call to all participants/clients (and only the right one will pick up it :-)

# Fix: The inspector sends a verification SMS to the SMS ticket sender

- the inspector sends a verification SMS to the SMS ticket sender (because he knows his number)
- **hacker's workaround:**
  - if the inspector's verification SMS message is received by the central SMS hack server, this SMS is redistributed to all participants/clients

# Fix: The inspector can ask the user to make a call to his number

- the inspector can ask the user to make a call to his predefined number (and he checks if the same sender number is used that was used for the SMS ticket request)
- **hacker's workaround:**
  - user can run his specific application that makes VOIP call through the central SMS hack server to the inspector phone – the inspector's will see the same number that was used for SMS ticket request

# Fix: The inspector can ask the user to send him a verification SMS

- the inspector can ask the user to send him a verification SMS (to verify if the user number is the same as the sender number that was used for the SMS ticket request)
- **hacker's workaround:**
  - user can run his specific application that sends this verification SMS message through the central SMS hack server, so the inspector will see the same sender number that was used for the SMS ticket request)

# Fix: Blacklist all numbers that are used by the central SMS hack server

- public transport service can reveal that many SMS ticket requests are sent from the same number. Consequently it can blacklist this number and reject all SMS tickets requests.
- **hacker's workaround:**
  - Change your SIM/mobile number periodically (SIM cards/unique numbers can be bought **COMPLETELY ANONYMOUSLY** in Czech Republic)
  - Use multiple SIMs/mobile numbers in the central SMS hack server
  - Use private P2P mobile networks

# Ahead to perfection!

- is to possible to spoof SMS sender or Caller ID?
- yes, of course - use public available services:
- http://phonytext.com
- http://www.telespoof.com
- http://www.spoofcard.com
- or use VOIP provider who allows you to set your own Caller-ID

# Private P2P mobile network

- **single point of detection** – the central SMS hack server uses the unique and still same phone number – this phone number can be easily detected and blacklisted
- every SMS hack client should have **own SMS sending server**
- when the central SMS hack server received **Get_Token_Request** or **Regenerate_Token_Request** from any client, it will choose a suitable client using fair scheduling algorithm that is consequently used for sending a real SMS ticket request, **Get_Token_Response** is returned to the central SMS server and then redistributed to all clients
- there is **no single point of detection** – every client participates in sending of SMS ticket requests – P2P mobile network cannot be easily blacklisted

# Using IMEI for passenger identification

- phone IMEI is a unique identifier that is not supposed to be a personal information (but in many countries it is illegal to change it!)
- IMEI can be easily checked by the inspector - he just types *#06# (it works on every phone)
- we can bind the passenger's IMEI with his SMS ticket (after his proper registration)
- when we have IMEI, we can do the passenger localization (with GSM operator support)

# Geographical localization of checked passengers

- probably the only way how to detect this attack
- we need to know the response for a question: **“Is a just checked passenger sufficiently closed (in a defined distance) to the inspector?**”
- cooperation with GSM operators is inevitable (they have to provide the name/position of the BTS the passenger/inspector is just associated with)

# Cooperation with GSM operators

- GSM operator will provide the binary function **IMEI_proximity(IMEI1, IMEI2)** that returns true if the mobile phone with IMEI1 (passenger) is closer to IMEI2 (inspector) than the defined distance
- to prevent misuse, only a subset of all inspector's mobile phone numbers can be used as IMEI2 and this function should be allowed only from inspector's PDAs (not to everyone)

# Issues regarding GSM localization

- it can be really difficult and expensive to implement **IMEI_proximity** function (cooperation of all GSM operators is required)
- the inspector needs to type IMEI of every checked passenger which is really time-consuming
- **Summary:** geographical localization of checked passengers can be very difficult and expensive

# Proposed security fix that works

- it is necessary to bind the user identity with his SMS ticket
- the user who wishes to use SMS tickets needs to REGISTER himself to the SMS service using SMS with text:
- REGISTER PERSONAL_USER_IDNUMBER or REGISTER PERSONAL_USER_BIRTHDATE

  (or REGISTER USER_MOBILEPHONE_IMEI)

# Proposed security fix II

- public transport server computes hash of the user's ID number or his birthdate and associates it with the user phone number (hash(PERSONAL_USER_IDNUMBER) <-> USER_PHONENUMBER)
- public transport server stores given pair to the central database
- plain-text personal information is not exposed (but be aware of rainbow tables attacks!)

# Firstly, let's explain some variables

- RAND_STRING - completely random string generated by the SMS server
- HASH_USERID – hash of the user personal ID
- ENC_RAND_STRING – RAND_STRING encrypted by the HASH_USERID
- DP_PRIVATE_KEY – public transport service private key
- DP_PUBLIC_KEY – public transport service public key
- DP_SYMETRIC_KEY – public transport service symmetric key
- SMS_TICKET_SYM – ENC_RAND_STRING encrypted by DP_SYMETRIC_KEY
- SMS_TICKET_ASYM – ENC_RAND_STRING encrypted by DP_PRIVATE_KEY

# Suitable encryption ciphers

- hashes – blowfish, ..
- symmetric encryption – AES, blowfish, twofish
- asymmetric encryption – ciphers that use short keys should be used (e.g. Elliptic curve cryptography – ECC)
- SMS ticket should be sufficiently short in order to speed up inspector's checking

# Secure and easy-usable solution

- **symmetric encryption is used** - the same key will be stored on the central SMS ticket server and distributed to all inspector's PDAs
- **advantage:** inspector does not need to write off user's SMS tickets, just scans user's personal ID and compares the PDA's result with the user SMS ticket
- **disadvantage:** a lost or stolen PDA can threat genuineness of SMS tickets

# SMS ticket generation using sym.key

1. the user asked SMS server for valid SMS ticket
2. Server generates completely random string (RAND_STRING)
3. This random string is symmetrically encrypted by HASH_USERID (that was looked up for given USER_NUMBER from the central database)
4. The result is encrypted by symmetric key DP_SYMETRIC_KEY and optionally hashed – final SMS_TICKET_SYM is gained

# SMS ticket generation using sym.encryption (mathematical) II

- ENC_RAND_STRING = AES(RAND_STRING, HASH_USERID)
- SMS_TICKET_SYM = HASH(AES(ENC_RAND_STRING), DP_SYMMETRIC_KEY)
- **SMS_TICKET_SYM = HASH(AES(AES(RAND_STRING, HASH_USERID)), DP_SYMMETRIC_KEY))**

# Verify process using sym.encryption

1. the user has to show the inspector his valid personal ID
2. the inspector's PDA scans his ID, computes its hash, discard its plain-text and makes look-up using HASH_USER_ID to the central SMS ticket server
3. if any valid SMS ticket has been already sent to the user (with given HASH_USER_ID), the server returns RAND_STRING, otherwise if the user has not asked for the SMS ticket or his SMS ticket has already expired, verification process will fail (this step should be sufficient to verify whether someone asked for the valid ticket encrypted by his HASH_USER_ID or not)
4. the inspector's PDA computes ENC_RAND_STRING by encrypting RAND_STRING using HASH_USER_ID, the result is encrypted by symmetrical key DP_SYMMETRIC_KEY and hashed. The final result is HASH(AES(AES(RAND_STRING, HASH_USER_ID), DP_SYMMETRIC_KEY)))
5. the inspector just compares if this value is the same as the user SMS ticket (by sight), if yes, the user uses valid SMS token that is associated with his identity

# Even more secure, but a bit complicated solution

- **asymmetric encryption (PKI) is used** – a private key will be stored on the central SMS ticket server only, public key is distributed to all inspector's PDAs
- **advantage:** a lost or stolen PDA cannot threat genuineness of SMS tickets
- **disadvantage:** inspector needs to write off to his PDA all SMS tickets he is checking (but this is the same for all current SMS ticket's implementations)

# SMS ticket generation using asym. encryption

1. the user asked SMS server for valid SMS ticket
2. Server generates completely random string (RAND_STRING)
3. This random string is symmetrically encrypted by HASH_USERID (that was looked up for given USER_NUMBER from the central database)
4. If the result is too long, it is hashed (optional phase)
5. The result is encrypted by private key DP_PRIVATE_KEY – final SMS_TICKET_ASYM is gained

# SMS ticket generation using asym.encryption (mathematical) II

- ENC_RAND_STRING = AES(RAND_STRING, HASH_USERID)
- SMS_TICKET_ASYM = ECC(HASH(ENC_RAND_STRING), DP_PRIVATE_KEY)
- **SMS_TICKET_ASYM = ECC(HASH(AES(RAND_STRING, HASH_USERID)), DP_PRIVATE_KEY)**

# Verify process using asym.encryption

1. the user has to show the inspector his valid personal ID
2. the inspector's PDA scans his ID, computes its hash, discard its plain-text and makes look-up using HASH_USER_ID to the central SMS ticket server
3. if any valid SMS ticket has been already sent to the user (with given HASH_USER_ID), the server returns RAND_STRING, otherwise if the user has not asked for the SMS ticket or his SMS ticket has already expired, verification process will fail (this step should be sufficient to verify whether someone asked for the valid ticket encrypted by his HASH_USER_ID or not)
4. The inspector's PDA computes ENC_RAND_STRING by encrypting RAND_STRING using HASH_USER_ID and computes its hash HASH(ENC_RAND_STRING)
5. The inspector types user's SMS ticket (SMS_TICKET_ASYM) to his PDA, due to the fact that this SMS ticket was encrypted using DP_PRIVATE_KEY, the inspector can decrypt it using his integrated DP_PUBLIC_KEY and gains HASH(ENC_RAND_STRING)
6. The inspector's PDA compares if computed hash from 4) paragraph is the same as computed hash from 5) paragraph, if yes, the user uses valid SMS token that is associated with his identity

# Does the fix look complicated?

- not really, the inspector just scans you personal ID document and visually compares the result of his PDA with your SMS ticket – that's all
- inspector's PDAs should support automatic personal ID scanning
- symmetric keys can be used even if the inspector's PDA is lost/stolen (it is necessary immediately to redistribute all new keys to all inspector's PDAs and central SMS server)

# Known public transport hacks

- London Tube Oyster
  http://www.cs.virginia.edu/~kn5f/Mifare.Cryptana
- MIT Students Hack Boston Public Transportation
  http://www-tech.mit.edu/V128/N30/subway/Defc

# Any questions?

# Thank you for listening

Ing. Pavol Lupták, CISSP CEH

# Appendix: SMS hack client/server commands

- a brief description of used SMS hack client/server commands for possible implementations is mentioned
- **Nethemba s.r.o.:**
- does not and will not release its own implementation (to prevent a public misuse)
- is not responsible for any existing functional implementations of the described hack

# Get_Token_Request

- **input parameters:**
  - User ID (the user's login/IP or CN from user's certificate)
- the request is sent by the user's SMS hack client when the user asks for a valid SMS ticket (by clicking to “Get SMS ticket” button in his application)

# Get_Token_Response

- SMS hack server checks if it has a valid SMS ticket, if yes, it sends the SMS ticket information back to the client in **Get_Token_Response**, if no, it asks for a new SMS ticket using connected mobile phone and subsequently sends the SMS ticket information back to the client in **Get_Token_Response**
- SMS hack server holds in its memory/database which SMS hack clients have valid SMS tickets and remove them if the SMS ticket expires

# Get_Token_Response II

- when the hack client receives **Get_Token_Response** it locally generates appropriate SMS ticket (with the same sender, recipient, body and status flag that were used in the original SMS ticket)
- there is **NO VISUAL DIFFERENCE** between the original SMS ticket and generated one
- there is no way for the inspector to reveal that you use invalid/locally generated SMS ticket

# Regenerate_Token_Request

- **input parameters:**
  - User ID (the user's login/IP or CN from user's certificate)
- the request is sent by user's SMS hack client when there is a need for a new SMS ticket (e.g. the user is stopped and checked by the inspector)
- it informs the central SMS hack server to invalidate existing SMS ticket and ask for a new valid one
- the central SMS hack server sends the new ticket to all connected SMS hack clients
- sending of this request can be fully automatized (if the SMS message “DPT OK” is received, Regenerate_Token_Request will be sent)

# Regenerate_Token_Response

- the **regenerate_token_response** is sent by the central SMS hack server to all connected SMS hack clients (those that have valid SMS tickets)
- should be sent in the same TCP connection that was used for **Get_Token_Request** (a TCP keep-alive traffic during given ticket's validity should be minimal)

# Get_Status_Request / Get_Status_Response

- **input parameters:**
  - User ID (the user's login/IP or CN from user's certificate)
- **Get_Status_Request** is sent by the SMS hack client to the central SMS hack server when the user wants to know if there is any valid SMS ticket (i.e. someone has already asked for it)
- if the SMS hack server has a valid SMS ticket, it sends information about it in **Get_Status_Response**, otherwise empty **Get_Status_Response** is sent

# Set_Options

- **input parameters:** type of parameter, parameter value
- **be_notified** yes/no (it is set by a SMS hack client in case it wants to be notified about presence of the valid SMS ticket on the SMS hack server)
- client periodically (e.g. every 15 minutes) sends **Get_Status_Request** and waits for **Get_Status_Response**
- if there is a valid ticket, the user will be notified

# Send_GEO_Position

- to avoid geographical collisions (passengers with the same ticket will meet each other in the same bus), mobile phone periodically sends its geographical position to the central hack SMS server:
- **GPS coordinates** (if the mobile phone has an integrated GPS)
- **BTS names/signals** (that is consequently used for triangulation and position approximation → in the same way Google Maps does it)