Mobile Terminal Security

Olivier Benoit[1], **Nora Dabbous**[2], **Laurent Gauteron**[1], **Pierre Girard**[1]

Helena Handschuh[2], **David Naccache**[2], **Stéphane Socié** [1], **Claire Whelan**[3]

1. Gemplus Innovation
La Vigie. Avenue des Jujubiers
La Ciotat, F-13705, France
{given_name.family_name}@gemplus.com

2. Gemplus Innovation
34 rue Guynemer
Issy-les-Moulineaux, F-92447, France
{given_name.family_name}@gemplus.com

3. Dublin City University
School of Computing
Glasnevin, Dublin 9, Ireland
cwhelan@computing.dcu.ie

1 Introduction

The miniaturization of electronics and recent developments in biometric and screen technologies will permit a pervasive presence of embedded systems. This - and the inclusion of networking capabilities and IP addresses in many handheld devices - will foster the widespread deployment of personal mobile equipment.

As mobile devices proliferate and their diversity grows, it is surprising to discover how few are appropriately secured against the risks associated with potential sensitive date exposure.

Mobile equipment fulfills a steadily growing variety of functions: holding personal data, interacting with other devices in local environment, communicating with remote systems, representing the person by making decisions, and processing data according to pre-established policies or by means of auto-learning procedures, to name a few.

From a software design perspective, modern mobile devices are real miniature computers embarking advanced software components linker, a loader, a Java virtual machine, remote method invocation modules, a bytecode verifier, a garbage collector, cryptographic libraries, a complex protocol stack plus numerous other specialized software and hardware components (*e.g.* a digital camera, a biometric sensor, wireless modems *etc.*).

Consequently, mobile devices need essentially the same types of security measures as enterprise networks – access control, user authentication, data encryption, a firewall, intrusion prevention and protection from malicious code.

However, the fundamental security difference inherent to mobile devices is the lack of physical access control. Mobile devices are designed for use outside the physical confines of the office or factory. Consequently, handheld devices and smart phones are often used precisely where they're most vulnerable – in public places, lobbies, taxis, airplanes – where risks include loss; probing or downloading of data by unauthorized persons; and frequently, theft and analysis of the device itself. Hence, in addition to logical security measures, mobile devices must embark protective mechanisms against *physical* attacks.

Note that inappropriate protection does not endanger only the mobile equipment but *the entire infrastructure*: mobile devices are increasingly Internet-connected as salespeople log on from hotel rooms and as field workers carry handheld devices with wireless networking. Of course, Internet activity exposes mobile devices to all the risks faced by an enterprise network including penetration and theft of important secrets. With fast processors and large memory, our mobile equipment carries current and critical data that may lead to financial loss if compromised. But the problem doesn't end there – these same devices generally also contain log-on scripts, passwords and user credentials that can be used to compromise the *company network itself* [1, 2].

This work attempts to overview these diverse aspects of mobile device security. We will describe mobile networks' security (WLAN and WPAN security, GSM and 3GPP security) and address platform security issues such as bytecode verification for mobile equipment and protection against viruses and Trojan horses in mobile environment - with a concrete J2ME implementation example. Finally we will turn to hardware attacks and briefly survey the physical weaknesses that can be exploited to compromise mobile equipment.

2 WLAN and WPAN security

When wireless communication protocols where first designed security wasn't among the primary goals. Most specifications included an optional basic protection for confidentiality, but weak algorithms were chosen for integrity and authentication. In the following subsections we will report security requirements and attacks in wireless local and personal area networks.

2.1 802.11 and Wi-Fi

The Wi-Fi alliance is a nonprofit international association formed in 1999 to certify interoperability of Wireless Local Area Network (WLAN) products based on the IEEE 802.11 specification. Since the first weaknesses in 802.11 communications were discovered, companies that wanted security relied on Virtual Private Networks (VPNs) rather than the wireless mean's security features. The Wi-Fi Alliance was concerned that lack of strong wireless security would hinder the use of Wi-Fi devices. For this reason in April 2003 it published the Wi-Fi Protected Access security requirements based on IEEE enhanced security draft status at that time [5].

2.1.1 802.11 security features

The only security services defined in the 802.11 original standard [3] were authentication and encryption. Key distribution had to be managed by the developer or the user and integrity was included for protection against transmission errors but not active attacks.

For authentication, Open System Authentication and Shared Key Authentication were supported. In both cases authentication could be replayed due to the lack of counters in packet transmission [12]. Moreover, Open System Authentication is a null authentication, successful whenever the recipient accepts to use this mode for authentication. A challenge response protocol was executed in Shared Key Authentication, but key distribution was not defined and the response was calculated based on WEP, the Wired Equivalent Protocol broken in 2001.

Initially WEP was the only algorithm designed for encryption. It is based on the stream cipher RC4, which outputs a key sequence given an initialization vector (IV) and a secret key as input. Ciphertext is obtained as the ex-or of the key sequence and the plaintext. Two key distribution schemes were defined, but for key mapping the key exchange between the source and destination station was out of the scope of the specification and when the default key system is used one out of 4 possible default keys must be chosen, greatly limiting the key space. In WEP there exists a large class of weak keys for which the first output bits can be easily determined. Moreover, because of the specific construction of the WEP key from a secret part and an initialization vector, if the same secret key is used with numerous different initialization vectors, an attacker can reconstruct the secret key with minimal effort [8], [9], [10]. Eavesdropping on a communication [11] is possible because initialization vector update is unspecified and often weak, and because wrap-around is many times neglected.

For integrity protection, a Checksum Redundancy Check was calculated. However CRCs don't allow detection of active attacks as they are non-keyed linear functions. Due to the weak integrity protection, a station can be thwarted to decrypt messages sent to a victim and redirect them towards the attacker[12].

2.1.2 802.11*i* security enhancements

In the year 2000, the 802.11*i* Working Group (WG) was created to enhance 802.11 security. The 802.11*i* standard was completed in June 2004.

802.11*i* working group main accomplishments concern the inclusion in the specification of strong authentication, secure encryption, addition of integrity protection mechanisms against active attacks and key generation and distribution.

For authentication, 802.11*i* WG decided to use 802.1*x* [4], a protocol initially developed for point to point wired communication but adaptable to wireless transmission as well. 802.1*x* defines end-to-end authentication between a station all the way to the authentication server using EAP methods. 802.1*x* also favors key distribution as after a successful authentication both ends, the station and the authentication server, share a secret key called Pairwise Master Key (PMK). Since wireless data exchange takes place between a station and an access point, 802.11*i* requests a 4-way authentication to occur after execution of the 802.1*x* protocol to verify the freshness of the communication between the station and the access point. The transfer of the PMK from the authentication server to the access point is out of the scope of 802.11*i*. Nevertheless, 802.11*i* defines a key hierarchy to derive encryption and integrity keys from the PMK.

802.11*i* supports 4 possibilities for encryption, that is no encryption, WEP, TKIP and CCMP. For each new encryption algorithm supported an integrity function was designed. When TKIP is chosen, integrity is obtained by using a Message Integrity Check (MIC) called Michael. CCMP provides simultaneously confidentiality and integrity.

TKIP and its related algorithm Michael were designed to solve problems encountered in WEP without requiring users to upgrade the hardware that grants them wireless connection. RC4 remains the core of TKIP, but a software modification in WLAN card MAC sections allows to address WEP weaknesses. Main modifications include use of longer initialization vectors, IV update on a per-packet basis and modification of the key mixing function. Michael is known to be vulnerable to brute force attacks, but it is the best compromise using legacy hardware. Countermeasures must be accounted for to reduce attacks on Michael.

CCMP requires a hardware update and should be used for maximum security. It is based on AES encryption algorithm used in counter mode for encryption. Integrity is provided by the calculation of a cipher block chaining message authentication code (CBC - MAC).

WPA supports 802.1x and pre-shared key authentication schemes. It supports both WEP and TKIP for data encryption, together with Michael for data integrity in the latter case. Key hierarchy is as defined in 802.11i draft 3.0. Wi-Fi Alliance will adopt the 802.11i final specification as WPA version 2. WPA is both backward and forward compatible: it is designed to run on existing Wi-Fi devices and should work with WPA2 devices as well.

2.2 802.15.1 and Bluetooth

In 1998, the Bluetooth Special Interest Group and IEEE 802.15.1 working group developed a technology for Wireless Personal Area Network (WPAN) communications.

The Bluetooth specification security features are based on secret key cryptographic algorithms. Authentication and encryption algorithms were specified, but no integrity protection was included.

Key generation functions and a challenge response mechanism for authentication are based on a 128-bit block cipher called SAFER-SK128. Until today, no weaknesses in SAFER have been published.

There are two possible ways to calculate the key that will be used by the devices for authentication, but the specifications state that using a device unit key for authentication purposes is insecure. A unit key is a semi-permanent key associated to a device, once it is disclosed device impersonation is possible for the lifetime of the unit key. Authentication based on device unit key was initially designed for constrained resource devices, and is maintained in the current specification for compatibility reasons. The authentication key should be computed as a combination key, that is a dynamic key whose value is determined by both peers and whose lifetime is generally shorter than that of a unit key.

Once the 128-bit encryption key is calculated, it is used to seed the stream cipher that generates the key sequence, with which the transmitted plaintext is ex-ored. Although an attack described in [6] demonstrates the reduction of the encryption key entropy space, the pre-computation effort to perform the attack is high enough to consider this attack of lesser relevance. Weaknesses in the cipher are also mentioned in [7], but the author himself defines the attacks not of practical relevance.

The main weakness in Bluetooth is in the paring mechanism, that is the procedure that allows two devices to share a same PIN. All Bluetooth keys, that is the initialization, authentication and encryption keys, are calculated based on the shared PIN. The PIN can be retrieved by performing a simple off-line attack and compromising the PIN leads to breaking Bluetooth's security. Since the PIN is the only secret in key generation and since generally 4 digit PIN codes are used, an attacker may find the PIN by recording a communication and exhaustively testing all 9999 possible PIN values. The attacker will know he's found the correct PIN when the calculated text sequence matches the recorded one.

Bluejacking is a much talked about security breach affecting Bluetooth communications. It involves sending a victim a message during the pairing phase. If the victim is thwarted into continuing the data exchange with the attacker until the handshake operation is concluded, pairing between the two devices will be obtained without the victim realizing it.

3 GSM and 3GPP security

The 3rd Generation Partnership Project (3GPP) is a follow-up project of the Global System for Mobile Communications (GSM). This third generation of mobile networks implements the UMTS (Universal Mobile Telecommunications System) standard. From a security perspective, 3GPP addresses a number of weaknesses and flaws in GSM and adds new features which allow to secure new services expected to be offered by UMTS networks [18].

3.1 GSM - Global System for Mobile Communications

GSM is one of the most widely used mobile telephone system. As communication with a mobile phone occurs over a radio link it is susceptible to attacks that passively monitor the airways (radio paths). The GSM specification addresses three key security requirements:

1. *Authentication* - To correctly identify the user for billing purposes and prevent fraudulent system use.
2. *Confidentiality* - To ensure that data (*i.e.* a conversation or SMS message) transmitted over the radio path is private.
3. *Anonymity* - To protect the caller's identity and location.

There are three proprietary algorithms used to achieve authentication and confidentiality. These are known as A3, A5 and A8. A3 is used to authenticate the SIM (Subscriber Identity Module) [1] for access to the network. A5 and A8 achieve confidentiality by scrambling the data sent across the airways. Anonymity is achieved by use of temporary identities (TMSI).

The process of authentication and confidentiality will now be explained in more detail. For a detailed account on the implementation of A3/5/8 we refer the reader to [19, 21].

3.1.1 Authentication

Authentication is achieved using a basic challenge-response mechanism between the SIM and the network. The actual A3 authentication algorithm used is the choice of the individual GSM network operators, although some parameters (input, output and key length) are specified so that interoperability can be achieved between different networks.

A3 is implemented in the SIM card and the Authentication Center (AuC) or the Home Local Register (HLR) [2]. A3 takes a 128 bit value Ki (subscriber i's specific authentication key) and 128 bit $RAND$ random number (challenge sent by the network) as input data. It produces a 32 bit output value $SRES$, which is a Signed RESponse to the networks challenge. The SIM and the network both have knowledge of Ki and the purpose of the authentication algorithm is for the SIM to prove knowledge of Ki in such a way that Ki is not disclosed. The SIM must respond correctly to the challenge to be authenticated and allowed access to the network. The authentication procedure is outlined in the following steps:

[1] The SIM associates the phone with a particular network. It contains the details (K_i and IMSI) necessary to access a particular account.

[2] HLR is a database that resides in a local wireless network. It contains service profiles and checks the identity of local subscribers.

1. The process is initiated by the user wanting to make a call from his mobile (Mobile Station or MS) or go on standby to receive calls.

2. The Visitor Location Register (VLR) [3] establishes the identity of the SIM. This is determined through a 5 digit temporary identity number known as the Temporary Mobile Subscriber Identity (TMSI). The TMSI is used in place of the International Mobile Subscriber Identity (IMSI). The IMSI is a unique number that identifies the subscriber worldwide. If the IMSI was used then this would enable an adversary to gain information about a subscribers details and location. The TMSI is frequently updated (every time the user moves to a new Location Area (LA) and/or after a certain time period) to stop an adversary from gaining such information. Note that there are situations where the IMSI will be used, for example on the first use of the mobile after purchase.

3. The VLR sends a request for authentication to the Home Location Register (HLR). This request will contain the SIM's IMSI (as the IMSI and the related TMSI should be stored in the VLR).

4. The HLR generates a 128 bit random $RAND$ challenge and sends it to the MS via the VLR.

5. Using Ki (128 bits) which is stored in the HLR and $RAND$ (128 bits), the HLR then calculates $SRES_{HLR}$ (32 bits) using the A3 authentication algorithm. $SRES_{HLR}$ is then sent to the VLR.

6. Using Ki (128 bits) which is stored in the SIM and $RAND$ (128 bits) that is received as a challenge, the SIM calculates $SRES_{SIM}$ (32 bits) using the A3 authentication algorithm. $SRES_{SIM}$ is then sent to the VLR.

7. If $SRES_{HLR} = SRES_{SIM}$, then the SIM is authenticated and allowed access to the network.

8. If $SRES_{HLR} \neq SRES_{SIM}$, an authentication rejected signal is sent to the SIM and access to the network is denied.

3.1.2 Confidentiality

Once the user has been successfully authenticated to the network, he can make calls and use the services he is subscribed to. It is necessary to encrypt the data that is transmitted over the airways, so that if it is intercepted, it will not be intelligible and in effect useless to an adversary.

The algorithm used to encrypt the data to be transmitted is called the ciphering algorithm A5. The key Kc used in this algorithm is generated by the cipher key generation algorithm A8. In a similar fashion to the A3 authentication algorithm, A8 takes $RAND$ and Ki and produces a 64 bit output value that is then used as the ciphering key Kc. A5 is a type of stream cipher that is implemented in the mobile station (MS) (as opposed to the SIM, where A3 and A8 are implemented). It takes Kc as input and produces a key stream KS as output. The key stream is ex-ored (modulo 2 addition) with the plaintext Pi, which is organised in 114 bit blocks. The resulting ciphertext block is then transmitted over the airways 114 bits at a time.

The process of authentication and enciphering is depicted in figure 1.

[3]The VLR is a network database that holds information about roaming wireless customers.

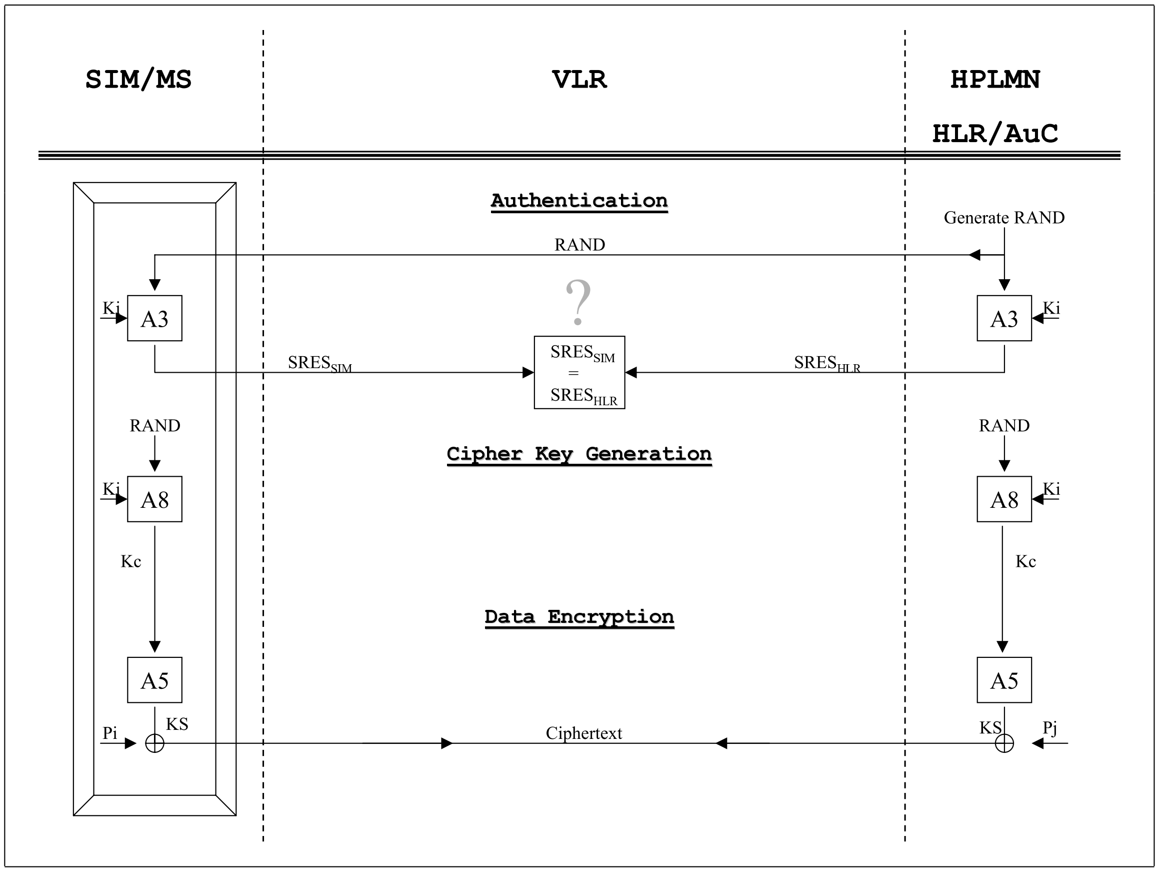


Figure 1: GSM Authentication and Ciphering

3.1.3 Limitations/Flaws of GSM

A number of weaknesses exist with GSM. One such flaw lies in the process of authentication. GSM only considers authentication as one way, *i.e.* the SIM authenticates itself to the network but the network does not authenticate itself to the SIM. This oversight enables an adversary to pretend to be a network by setting up a false base station with the same Mobile Network Code as the subscribers network. The adversary is thus in a position to engage in illegal interaction with the subscriber. Additionally the adversary can also partake in a man in the middle attack.

GSM only provides access security; it does not protect against active attacks. To give a few examples, user traffic and signalling information within the networks is done in clear text. In other words, except for the radio channel (*i.e.* the channel between the mobile equipment and the base station) data and voice encryption is turned off. Thus in particular, cipher keys and authentication tokens are sent in clear over the network, so that calls can be intercepted and users or network elements can be impersonated.

Another weakness with GSM lies in a particular implementation of the A3/A8 authentication [4] and cipher key generation algorithm COMP128. COMP128 is a type of keyed hash function. It takes 128 bit key and 128 bit random number as input (*Ki* and *RAND* as before), and produces a 96 bit digest as output. The first 32 bits are used as a response (*SRES*) back to

[4]A3 and A8 are implemented as one algorithm, namely COMP128.

the network's request for authentication. The remaining 64 bits are used as the session key (*Kc*) for voice encryption using the A5 algorithm. The first main flaw with COMP128 is that it was a proprietary encryption system developed behind closed doors. The problem with this kind of approach is that the algorithm is never subject to public scrutiny and so vulnerabilities and possible design flaws in the protocol are not given the opportunity to be identified. The proof of this is the fact that COMP128 has been cryptanalysed and reversed engineered [24]. Since the COMP128 algorithm was exposed a number of weaknesses have been found. One such weakness is that it is susceptible to a collision attack. This attack plays on a weakness in the second round of the algorithm that allows using carefully chosen *RAND* values (approximately 2^{17})[5] to determine *Ki*. COMP128 is also vulnerable to a type of power analysis attack [20] known as a partition attack [22]. This type of attack is a form of side channel attack that manipulates information that leaks naturally[6] from the SIM during its operation. The part of COMP128 that this attack exploits is in the table look up operations. COMP128 consists of 8 rounds, where each round consists of 5 levels of table look-up. The five look-up operations are performed modulo 512, 256, 128, 64 and 32 respectively. COMP128 is optimized for 8 bit processors by operating on one byte at a time. However, in the first look-up operation a 9 bit value is required to be accessed (modulo 512). This requires that the 9 bit value be split into two 8 bit values. This split can then be identified as a correlation between the power consumption and the internal instruction that the SIM is performing and effectively identify a number of key bits. By recursively repeating this process the key *Ki* can be reconstructed and recovered. This attack only requires 8 chosen plaintext values (*RAND*) and can be performed in a matter of minutes. Once an adversary is in possession of *Ki* he is capable of cloning [24] the SIM and can take on a person's identity and illegally bill his account.

Some of the flaws just described can be combined to perform an extremely destructive attack known as over the air cracking. Firstly an adversary imitates a legitimate GSM network. The mobile phone is paged by its TMSI to establish a radio connection. Once the connection is established, the attacker sends a request for the IMSI (this is within the right of a "legitimate" network). The attacker can then keep challenging the MS with carefully chosen *RANDs* so as to exploit flaws in the COMP128 algorithm. To each *RAND* the mobile phone will respond with a different *SRES*, which the attacker will collect and store. This process will be repeated until the attacker has gained enough information to learn *Ki*. Now the attacker has *Ki* and IMSI in their possession. This enables an attacker to impersonate the user, and make and receive calls and SMS messages in their name. They can also eavesdrop, since *RANDs* from the legitimate network to the legitimate user can be monitored, and thus combined with the known *Ki* can be used to determine the *Kc* used for voice and signaling data encryption. An intelligence expert confirmed that this procedure was effectively and regularly used by at least one intelligence service during the past decade.

Last but not least, GSM networks lack the flexibility to quickly upgrade and improve security elements such as the cryptographic algorithms. For instance, the encryption algorithm A5/3 and the authentication and key generation algorithm GSM-MILENAGE are already available, but have not been widely deployed yet.

This section mentions the most serious weaknesses with GSM, we refer the reader to [21, 23] for more details on attacks. These shortcomings have enabled a number of powerful and successful attacks to be made against GSM. The experience gained from isolating and rectifying

[5]Compared to a brute force attack that requires testing 2^{128} values for K.

[6]Timing, power consumption and electromagnetic emanations are types of side information that leak naturally form the SIM if proper countermeasures are not implemented.

these weaknesses have contributed to the evolution of a more secure mobile telephone technology 3GPP.

3.2 3GPP - 3rd Generation Partnership Project

3GPP specifications address both access security implementing mutual user and network authentication, and network security with strong user data, voice, and signalling data encryption and authentication.

3.2.1 Authentication and Key Agreement Protocol

The basic building block of 3GPP Security is its authentication and key agreement protocol (AKA) [13, 14]. Improving over GSM networks, UMTS networks provide over-the-air mutual authentication of the user to the network and of the network to the user, but also strong data and voice encryption and signalling data authentication between the mobile equipment and the radio network controller. In order to achieve these objectives, a similar approach to GSM is adopted. The telecommunications operator provides the end user with personal security credentials (*i.e.* an identity and a secret key), contained in a so-called USIM (User Services Identity Module), which in most cases takes the form of a smart card inserted into the mobile station (or MS). This USIM holds in particular a secret key (K) shared with the Authentication Center AuC of the operator; using this secret key and the AKA protocol, authentication tokens and encryption keys are derived by the USIM from a random challenge (RAND) sent by the network to the mobile equipment. Mutual authentication is achieved by a challenge response protocol in which the USIM receives the authentication token which allows it to check whether the network is genuine, and has to compute an authentication response RES (to be compared to the expected value $XRES$) for the network to gain access. The USIM also generates ciphering (CK) and integrity keys (IK) and makes them available to the mobile terminal. In addition, the network has to send a fresh sequence number (SQN), which provides evidence that the session keys and authentication tokens have not been used before and will not be used again. These sequence numbers have to remain within a certain range from previous sequence numbers in order to be considered valid. If at some point a sequence number is out of range, a special re-synchronization procedure enables to securely reset the sequence numbers and to take up new calls. An authentication management field allows the network to define which algorithms are used in which security function. Finally, an anonymity key (AK) is optionally used to conceal the sequence numbers – and therefore the identity of the subscriber – from an opponent. In figure 2, we provide a graphical overview of the procedure for generating authentication vectors (AV) in the basic AKA protocol. The example algorithm set for implementing security functions f1 to f5 in 3GPP networks is called MILENAGE [17].

3.2.2 Network Security

Once the user is authenticated to the network and access security is guaranteed, user data and signalling messages need to be protected in the network. A first phase of encryption and integrity checking is performed between the mobile terminal and the radio network controller on the radio link up to the security node. Encryption and data integrity computations are performed by the mobile equipment itself using one-time session keys derived by the USIM from the network challenge, UMTS encryption function f8 and integrity function f9, both standardized algorithms based on the block cipher KASUMI [15]. The function f8 may be used for encrypting user data

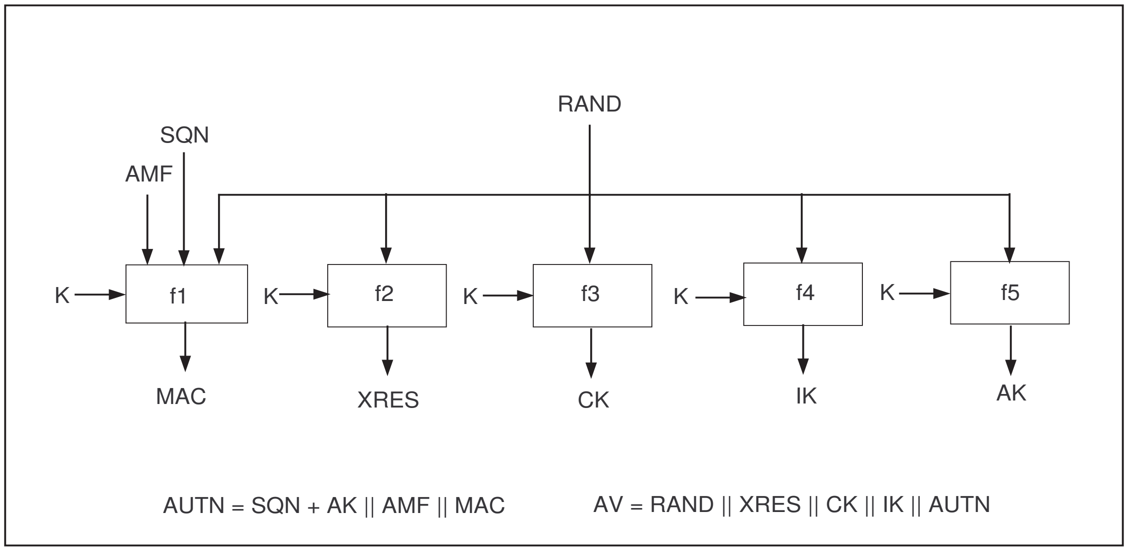

RAND
SQN
AMF
K
f1
K
f2
K
f3
K
f4
K
f5
MAC
XRES
CK
IK
AK
AUTN = SQN + AK || AMF || MAC
AV = RAND || XRES || CK || IK || AUTN

Figure 2: Authentication vector generation

as well as signalling messages between the mobile terminal and the radio network controller, whereas function f9 is only meant for integrity of signalling messages. In order to avoid the re-use of keystream and message authentication codes, both f8 and f9 use a time-dependent parameter COUNT. f8 also takes into account the bearer identity and manages the direction of the transmission with a DIRECTION field. f9 uses an additional fresh random value provided by the network to generate each new MAC.

Subsequently, a second phase of message encryption and authentication is provided directly within the global network between different operators and within the networks of the operators. A global public key infrastructure allows the Key Administration Center of each network to generate a public key pair and to store public keys from other networks, exchanged as part of the roaming agreements. Each Key Administration Center can then generate shared session keys and distribute these keys to different network entities within its own network, as well as to the Key Administration Center of another network, which in turn distributes the same shared session keys to its own network entities. These session keys are then used with standard symmetric encryption and data authentication algorithms within the networks.

This feature completes the second evolution with respect to GSM networks, for which no encryption of signalling messages and user traffic is available. All cryptographic algorithms mentioned in the context of 3GPP have been evaluated and are publicly available.

4 Mobile Platform Layer Security

Mobile terminals run a variety of operating systems, which, for most of them, are proprietary and remain hidden for the end user. In the hight end segment of the terminal market, the operating systems are no longer buried in the hardware and the consumer can choose between Symbian, PalmOS and Windows Mobile. However, these so-called smart terminals represent a small fraction of the deployed equipments. For the vast remaining majority the only way to download and execute software is to target the mobile edition of the Java Virtual Machine (aka as J2ME/CLDC/MIDP or MIDP for short) that is generally provided. Consequently, this section is entirely focused on the Java environment for mobile devices.

4.1 Bytecode Verification for Mobile Equipment

The Java architectures for mobile equipments [26] allow new applications, called *applets*, to be downloaded into mobile devices. While bringing considerable flexibility and extending the horizons of mobile equipment usage this *post issuance* feature raises major security issues. Upon their loading, malicious applets can try to subvert the Java Virtual Machine's (JVM) security in a variety of ways. For example, they might try to overflow the stack, hoping to modify memory locations which they are not allowed to access, cast objects inappropriately to corrupt arbitrary memory areas or even modify other programs (Trojan horse attacks). While the general security issues raised by applet download are well known [35], transferring Java's safety model into resource-constrained mobile devices such as smart-cards, handsets or PDAs appears to require the devising of delicate security-performance trade-offs.

When a Java class comes from a distrusted source, there is a way to ensure that no harm will be done by running it. The method consists in running the newly downloaded code in a completely protected environment (*sandbox*). Java's security model is based on sandboxes. The sandbox is a neutralization layer preventing access to unauthorized resources (hardware and/or software). In this model, applets are not compiled to machine language, but rather to a virtual-machine assembly-language called *byte-code*.

In a JVM, the sandbox relies on access control. Nevertheless an ill-formed class file could be able to bypass it. Therefore, there are two basic manners to check the correctness of a loaded class file.

The first is to interpret the code *defensively* [27]. A *defensive interpreter* is a JVM with built-in dynamic runtime verification capabilities. Defensive interpreters have the advantage of being able to run standard class files resulting from *any* Java compilation chain but appear to be slow: the security tests performed during interpretation slow-down each and every execution of the downloaded code and the memory complexity of these tests is not negligible either. This renders defensive interpreters relatively unattractive for mobile equipments where resources are severely constrained and where, in general, applets are downloaded rarely and run frequently.

Another method consists in a static analysis of the applet's byte-code called *byte-code verification*, the purpose of which is to make sure that the applet's code is well-typed to detect stack over/underflow, ... This is necessary to ascertain that the code will not attempt to violate Java's security policy by performing ill-typed operations at runtime, or by changing some system data. (*e.g.* forging object references from integers or calling directly API private methods). Today's *de facto* verification standard is Sun's algorithm [33].

In the rest of this section we recall Java's security model and the cost of running Sun's verification, and we briefly overview mobile-equipment-oriented alternatives to Sun's algorithm.

4.2 Java Security

The *Java Virtual Machine (JVM) Specification* [33] defines the executable file structure, called the *class file* format, to which all Java programs are compiled. In a class file, the executable code of *methods* (Java methods are the equivalent of C functions) is found in *code-array* structures. The executable code and some method-specific runtime information (namely, the maximal operand stack size $S_{\max}$ and the number of local variables $L_{\max}$ claimed by the method) constitute a *code-attribute*. We briefly overview the general stages that Java code goes through upon download.

To begin with, the classes of a Java program are translated into independent class files at compile-time. Upon a load request, a class file is transferred over the network to its recipient where, at link-time, symbolic references are resolved. Finally, upon method invocation, the relevant method code is interpreted (run) by the JVM.

Java's security model is enforced by the *class loader* restricting what can be loaded, the *class file verifier* guaranteeing the safety of the loaded code and the *security manager* and *access controller* restricting library methods calls so as to comply with the security policy. Class loading and security management are essentially an association of lookup tables and digital signatures and hence do not pose particular implementation problems. Byte-code verification, on which we focus this section, aims at predicting the runtime behavior of a method precisely enough to guarantee its safety without actually having to run it.

4.2.1 Byte-Code Verification

Byte-code verification [30] is a load-time phase where the method's run-time behavior is proved to be *semantically correct*.

The *byte-code* is the executable sequence of bytes of the code-array of a method's code-attribute. The byte-code verifier processes units of method-code stored as class file attributes. An initial byte-code verification pass breaks the byte sequence into successive instructions, recording the offset (*program point*) of each instruction. Some static constraints are checked to ensure that the byte-code sequence can be interpreted as a valid sequence of instructions taking the right number of arguments.

As this ends normally, the receiver assumes that the analyzed file complies with the general syntactical description of the class file format.

Then, a second verification step ascertains that the code will only manipulate values which types are compatible with Java's safety rules. This is achieved by a type-based *data-flow analysis* which abstractly executes the method's byte-code, by modelling the effect of the successive byte-codes on the *types* of the variables read or written by the code.

4.2.2 The Semantics of Type Checking

A natural way to analyze the behavior of a program is to study its effect on the machine's memory. At runtime, each program point can be looked upon as a memory *instruction frame* describing the set of all the runtime values possibly taken by the JVM's stack and local variables.

Since run-time information, such as actual input data is unknown before execution starts, the best an analysis may do is reason about *sets* of possible computations. An essential notion used for doing so is the *collecting semantics* defined in [28] where, instead of computing on a full *semantic* domain (values), one computes on a restricted *abstract* domain (types).

For reasoning with types, one must precisely classify the information expressed by types. A natural way to determine how (in)comparable types are is to rank all types in a *lattice* $\mathcal{L}$. A brief look at the toy lattice depicted below suffices to find-out that `animal` is more general than `fly`, that `int` and `spider` are not comparable and that `cat` is a specific `animal`. Hence, knowing that a variable is designed to safely contain an `animal`, one can infer that no harm can occur if during execution this variable would successively contain a `cat`, a `fly` and an `insect`. However, should the opposite be detected (*e.g.* an instruction would attempt to use a variable supposed to contain an `animal` as if it were a `cat`) the program should be rejected as unsafe.

The most general type is called *top* and denoted $\top$. $\top$ represents the *potential simultaneous presence of all types*, *i.e.* the *absence of (specific) information*. By definition, a special null-pointer type (denoted `null`) terminates the inheritance chain of all object descendants.

Formally, this defines a pointed complete partial order (CPO) $\preceq$ on the lattice $\mathcal{L}$.

$$\begin{tikzcd}
 & \top \arrow[dl] \arrow[dr] & & & \\
\texttt{int} \arrow[d] & & \texttt{Object} \arrow[d] & & \\
\texttt{null} & & \texttt{animal} \arrow[dl] \arrow[dr] & & \\
 & \texttt{cat} \arrow[d] & & \texttt{insect} \arrow[dl] \arrow[d] \arrow[dr] & \\
 & \texttt{null} & \texttt{spider} \arrow[d] & \texttt{bee} \arrow[d] & \texttt{fly} \arrow[d] \\
 & & \texttt{null} & \texttt{null} & \texttt{null}
\end{tikzcd}$$

Stack elements and local variable types are hence tuples of elements of $\mathcal{L}$ to which one can apply *point-wise ordering*.

The verification process described in [33] §4.9, is an (iterative data-flow analysis) *abstract interpretation* algorithm that attempts to build an *abstract description* of the JVM's memory for each program point. A byte-code is safe if the construction of such an abstract description succeeds.

Denoting by N_{blocks} the number of branches in a method, a straightforward implementation of [33] §4.9 allocates N_{blocks} frames, each of size $L_{\max} + S_{\max}$.

$L_{\max}$ and $S_{\max}$ are determined by the compiler and appear in the method's header. This results in an $\mathcal{O}((L_{\max} + S_{\max}) \times N_{\text{blocks}})$ memory-complexity. In practice, the verification of moderately complex methods would frequently require a few thousands of bytes.

4.2.3 Memory Economic Verification Approaches for Mobile Equipments

While the time and space complexities of this algorithm suit personal computers, the memory complexity of Sun's algorithm appears unadapted for mobile devices, where RAM is a significant cost-factor.

This limitation gave birth to a number of innovating workarounds where, in each case, memory was reduced at the expense of another system resource (transmission, computation *etc.*) or by transforming Sun's standard class file format to render it easier to verify:

- Leroy [31, 32] devised a verification scheme that relies on off-card code transformations whose purpose is to facilitate on-card verification by eliminating the memory-consuming fix-point calculations of Sun's original algorithm.

- *Proof carrying code* [37] (PCC) is a technique by which a side product of the verification, namely the final type information inferred at the end of the verification process (*fix-point*), is sent along with the byte-code to allow a straight-line verification of the applet. This extra information causes some transmission overhead, but the memory needed to verify a code becomes essentially equal to the RAM necessary to run it. A PCC off-card proof-generator is a rather complex software.

- *Variable-wise verification* [34] is a technique where variables are verified in turn rather than in parallel, re-using the same RAM space. This trades-off computations for memory.

- *Externalization* [29] consists in securely exporting intermediate verification variables to distrusted terminals. This trades-off transmission for memory.

We refer the reader to the bibliography for a more detailed information on these techniques.

4.3 Troyan Horses in Mobile Environment

A Trojan horse is a malevolent piece of code hidden in a program that performs normal tasks. When started, this program behaves as expected by the user and then stealthily executes the Trojan horse payload. Popular games and sharewares, especially if they are downloaded from the Internet are good vectors for Trojan horses.

Worms, which are self-propagating pieces of malicious software who propagate from one computer to another via a network link, have become very common in the past few years on PC even if their payloads have often been non-destructive. The first worm for smart phone showed-up recently targeting Symbian terminals and propagating itself via Bluetooth links [39]. Java Virtual machines are immune, by design, to this kind of attacks, so we will only discuss Trojan horses in the following.

The ultimate goal of a Trojan horse can just be a denial of service or a hacker's demonstration of power as in most of currently existing worms and viruses in the PC world. But some attractive targets can motivate an attacker on a mobile equipment. Nowadays these devices are fully merged in our life-style and they abound in credentials, personals information like contacts or to-do lists, let alone our real time position on the earth.

To demonstrate the potential wrongdoing and stealthiness of a Trojan horse we have implemented a prototype on a mainstream GSM phone. We have taken advantage of the fact that a java application for the J2ME/CLDC/MIDP environment (a MIDlet) is capable of taking the full graphic control of the handset screen, *i.e.* the programmer can control each and every pixel of the screen surface. The consequence is that a MIDlet can mimic the look-and-feel of any application including the system ones. In our example, the Trojan horse is lurking in a popular game called Tic Tac Toe and is aimed at capturing the SIM card's PIN that is entered by the user when the phone is switched-on. Figure 3 shows the general scheme of the attack.

When the game is started for the first time the Trojan horse is activated and simulates a phone reboot, including the vendor's logo animation and the PIN entry. This phase is unlikely to alert the average user that something is going wrong as she's used to such reboots due to battery shortage or software instability. The Trojan horse captures the user's PIN and terminates the MIDlet. This first phase is illustrated by the screen shots in figure 4.

In the subsequent MIDlet launches, the Trojan horse keeps quiet and the user is able to play with a genuine looking game. Nevertheless, the Trojan horse is still waiting for a backdoor code that reactivates it in order to display the PIN previously captured as shown in figure 5.

The lesson learnt from this school case example is that the mobile phone lacks from a trusted path between the user and the phone operating system both for input and output. In other

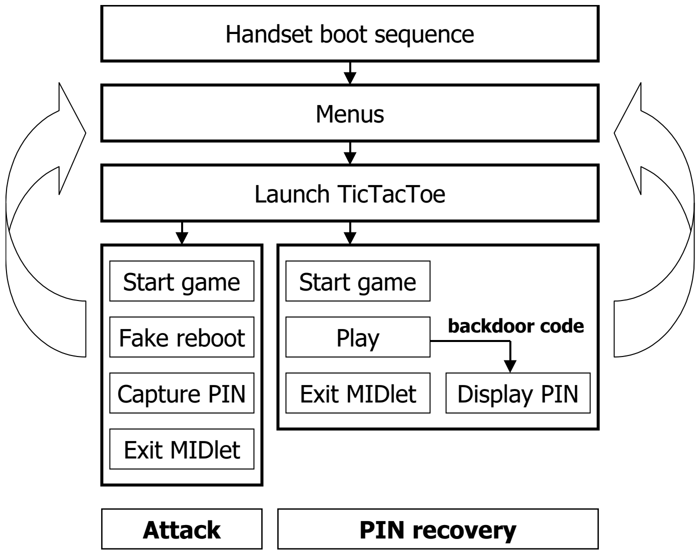


Figure 3: General scheme of a MIDlet Trojan horse

words there is no mean for the user to know if she communicates with the operating system or a malicious software which impersonates it.

One possible solution would be to limit the screen area that a MIDlet can control and to dedicate the remaining part to the OS that could use it to draw the user's attention on the fact that a MIDlet is running. Concerning the input part of the problem a dedicated key can be pressed before entering the PIN code in order to switch to the Operating System if it was not the foreground task. The problem with these solutions depicted on figure 6 is that they restrain further the restricted hardware available for the developers.

5 Hardware Attacks on Mobile Equipments

The term "hardware attack" encompasses a large variety of threats that exist because of the physical properties of the device under consideration. As a consequence of this definition a virtual design is not subject to such attacks and by extension a device physically out of the attacker's reach is also safe. By contrast, software attacks are most of the time remote attacks on a device attached to a network but physically out of the hacker's reach.

There are different ways to classify hardware attacks, among which is their belonging to one of the following categories: invasive attacks, fault attacks or side-channel analysis. A device designed to resist the above listed threats is called "tamper-resistant". In other words, a tamper resistant device will withstand attempts to tamper with the device (recover information or modify internal data or any characteristics of the device). Another feature that a device might exhibit is "tamper-evidence", signifying that evidence will exist to prove tampering with the device. At present, the only existent tamper-resistant element in a handset is the (U)SIM (Universal Subscriber Identity Module), where tamper resistance is achieved by the appropriate combination of hardware and software protection, counter-measures and prudent design rules.

The following paragraphs will provide an overview of handset attack targets before showing how to perform physical attacks and describing what benefits a hacker might gain.

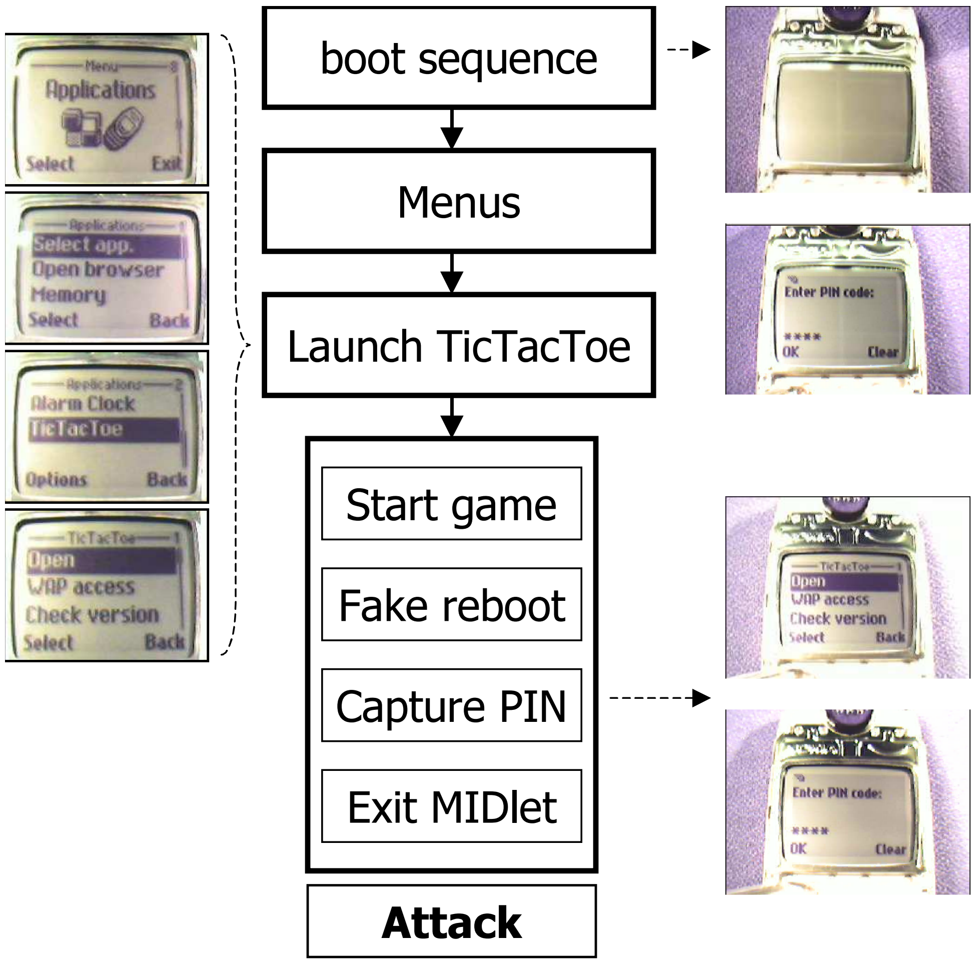


Figure 4: Attack phase of a MIDlet Trojan horse

5.1 Attack Targets

Secret or sensitive data is usually the target of an attack. Secret data is unknown by the hacker and his primary goal is to retrieve its value. Sensitive data is known by the hacker but cannot be modified by him; his primary goal is to modify its value, preferably to replace it with a value of his choosing. There are currently several targets in mobile equipments. The most sensitive data elements are the user authentication key (Ki), his identification number IMSI and the (CHV) (Card Holder Verification) value. In addition, there are at least 3 relevant targets in the handset: the SIM-lock mechanism, the IMEI and the software upgrade. Each of these targets is addressed hereafter.

5.1.1 SIM-Lock

SIM-lock is a mechanism commonly used by Mobile Network Operators (MNOs) to bind subsidized phones to the network [16], at least for a specified period of time. Such a binding should usually last until the operator's initial investment has been recouped. Nevertheless, if the subscriber wants to use a different network before the specified period of time is over, he needs to de-SIMlock his mobile. This service is not free, MNOs usually request around 115 euros to

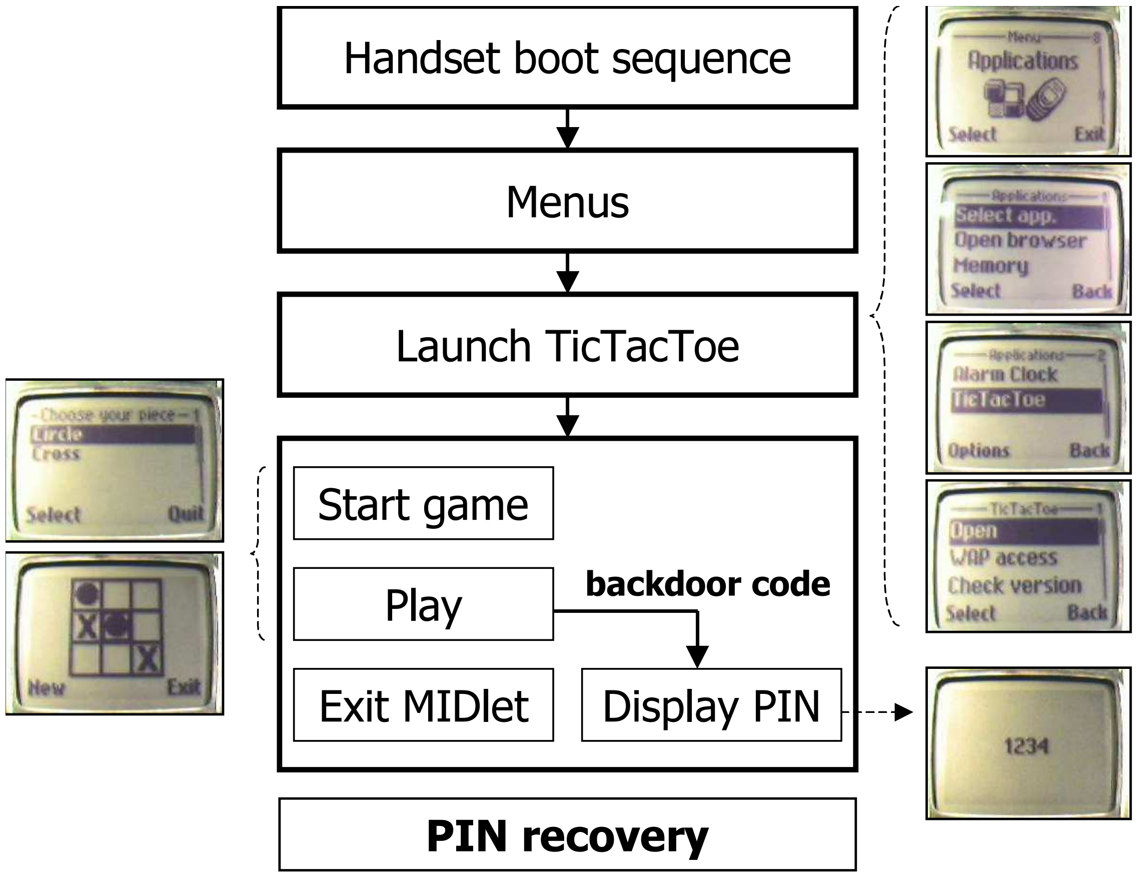


Figure 5: PIN recovery phase of a MIDlet Trojan horse

unlock a mobile phone. The very lucrative business coming from stolen handsets is slightly hindered by the SIM-lock mechanism. Indeed, the handset must be unlocked prior to usage by its new owner. As it is not illegal to unlock a phone, some software companies entered this business and provide unlocking software. An example of such software GUI (Graphic User Interface) can be seen on Figure 7.

5.1.2 IMEI

The International Mobile Equipment Identity number is the identity of the handset. It is a unique number attributed during handset manufacturing and is registered by the Mobile Network Operator. Thanks to IMEI, Mobile Equipment declared as stolen can be black-listed by the MNOs. Nevertheless, there is currently no IMEI blacklist at a worldwide level, stolen phones often leave their original country for less developed countries where people cannot afford the price of a new handset. To use the handset in the same country it has been stolen in, the IMEI value can also be changed to an authorized one. Some countries have voted laws that make IMEI alteration illegal to reduce handset theft. In parallel, handset manufacturers are working on increasing the IMEI's security.

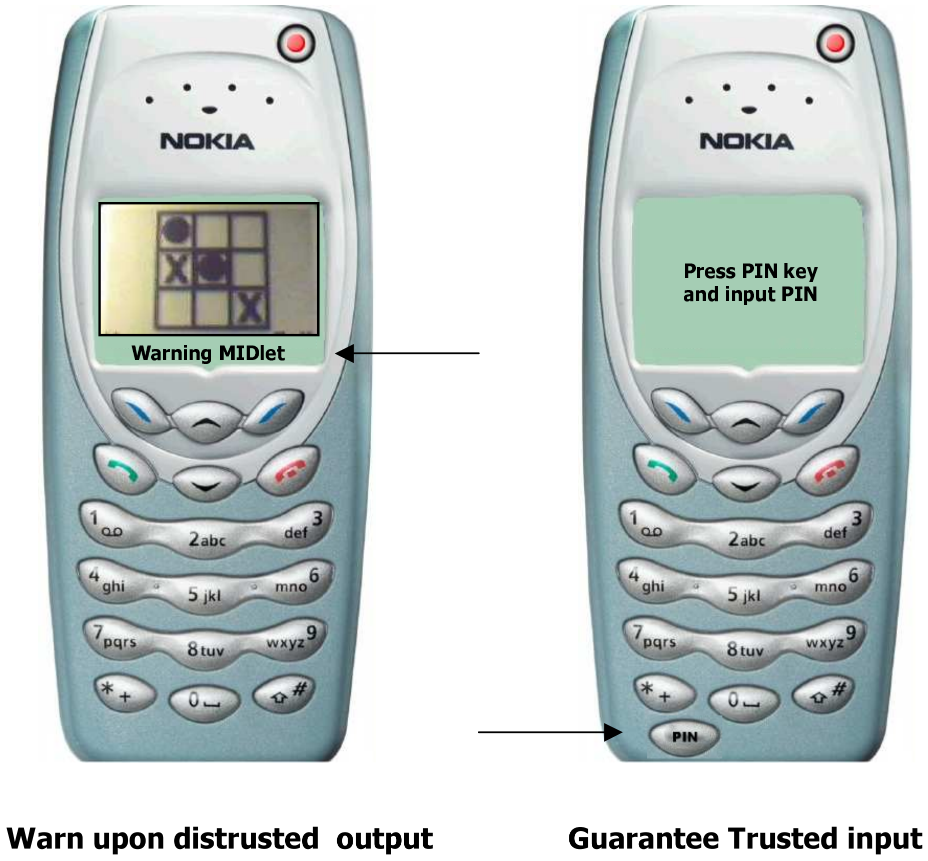


Figure 6: Trusted path on GSM phone

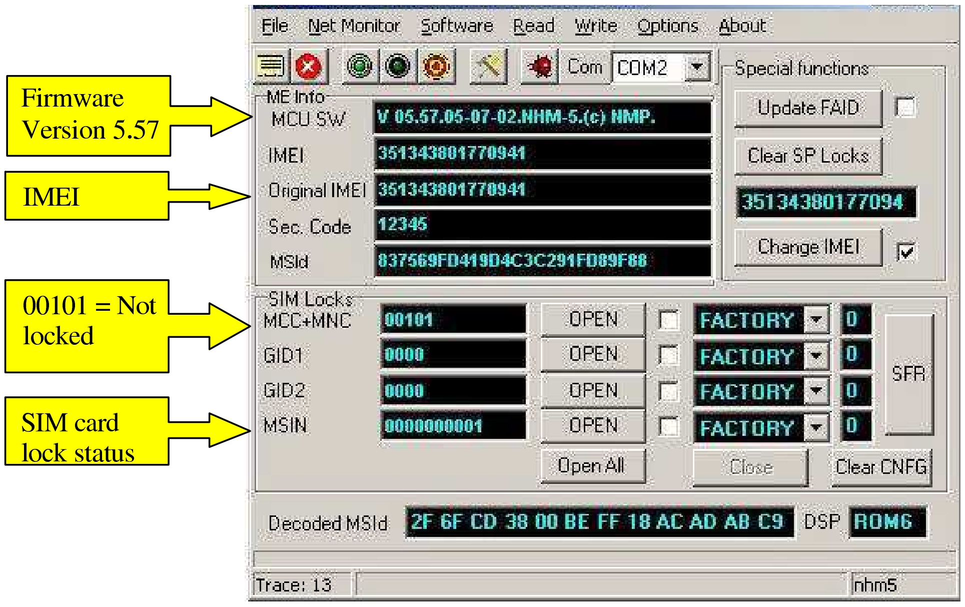
5.1.3 Software Versions

For a given mobile equipment, multiple software and firmware versions are available. High end versions usually add extra features and functionalities, making it lucrative for a hacker to upgrade a software version to a higher one. The upgrade mechanism is currently slightly protected, against unauthorized access depending on the handset model.

5.2 Hardware Attacks Description

Currently, handsets are in such a poor security state that they do not withstand basic reverse engineering weaponry. Moreover, their security mechanism such as the SIMlock, test/debug mode, IMEI storage and software upgrade are poorly designed and rely on obscurity rather than strong cryptographic protocols. Breaking these mechanisms does not yet require use of advanced attack techniques such as hardware attacks, which are at routinely researched in the industry and university research labs.

Fortunately, mobile equipment and chipset manufacturers are working hard to cath-up and improve the overall security level of handsets. As security will increases and software attacks will become less practical, hardware attacks will rise.



Com COM2

ME Info
MCU SW V 05.57.05-07-02.NHM-5.(c) NMP.
IMEI 351343801770941
Original IMEI 351343801770941
Sec. Code 12345
MSId 837569FD419D4C3C291FD89F88

35134380177094

SIM Locks

MCC+MNC	00101	OPEN	☐	FACTORY	0
GID1	0000	OPEN	☐	FACTORY	0
GID2	0000	OPEN	☐	FACTORY	0
MSIN	0000000001	OPEN	☐	FACTORY	0

Decoded MSId 2F 6F CD 38 00 BE FF 18 AC AD AB C9 DSP ROM6

Trace: 13 nhm5

Figure 7: Unlocking software interface.

5.2.1 Invasive Attacks

Invasive attacks are usually considered as the heaviest class of attacks in terms of equipment cost, expertise and duration. An invasive attack requires first of all to ”open” the device. This is not an easy task on a smart card as delicate chemistry manipulation is needed. On the other hand, on a handset only removal of the plastic case and eventually a few screws is required. In a smartcard such as a USIM, resistance against invasive attacks is achieved by embedding the complete system, including the CPU (Central Processing Unit), memories and peripherals, in a single chip. Moreover, the design usually includes additional security features such as protection shields, glue logic design, encryption and scrambling. Such architecture will probably not reach the handset field because combining different technologies such as a CPU, a large Flash memory and a RAM (Random Access Memory) memory on the same chip highly increases its cost. In a regular handset, the SoC (System On Chip) comprising the CPU and some peripherals, as well as the external memory (usually a flash containing both the operating system and the users personal data) can be found on same PCB (Printed Circuit Board). With such architecture, it is currently quite easy to probe the bus between the SoC and the Flash in order to gain access to all the data accessed by the CPU. This is a straightforward way gain access to secret information stored in the Flash (IMEI, unblock code). Of course it will require a little bit of reverse engineering and electronic skills since the data bus is usually 16 to 32 bits wide and since most of the lines will be buried in the internal layers of the multi-layer PCB. Another invasive attack consists in de-soldering the Flash memory chip in order to reprogram it with a flash programming unit or to replace it with a new Flash. Such an operation is not possible with a regular soldering iron because Flash memory packaging is usually of TFBGA (Thin & Fine-pitch Ball Grid Array) type. A printed circuit board from mobile equipment with its Flash memory removed can be seen Figure 8.a. The backside of a TFBGA Flash memory is shown Figure 8.b. Last but not least, most handsets provide a JTAG bus or others facilities for debug and test mode. This is a prime backdoor because with a JTAG cable and a little bit of insider knowledge a hacker can easily access very sensitive and secret information and do almost whatever he wants on a handset. There is no such threat on smart-cards since the debug and test mode is completely wiped-out at the end of the manufacturing process, usually by placing the corresponding logic on the scribe line of the wafer.

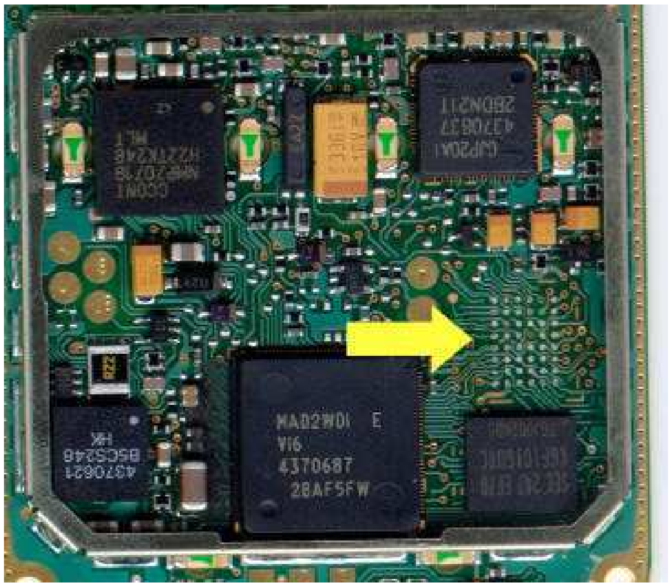


Figure 8.a: Circuit with Flash memory removed

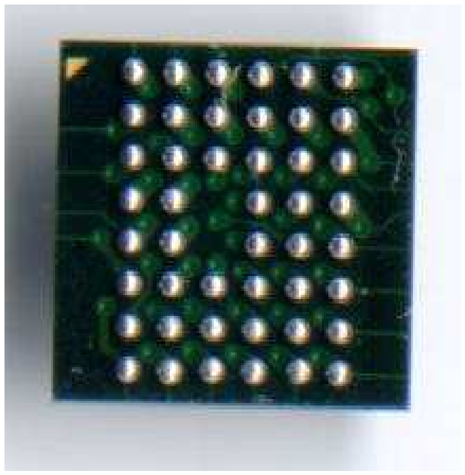


Figure 8.b: Flash desoldered & reballed

5.2.2 Side-Channel Attacks

Side-channel attacks consist in monitoring a device signal or resource-consumption, usually without physically damaging it. The processing's duration, power consumption, electro-magnetic radiations and radio-frequency emission are typically the signals that might be of interest. Once the signal has been monitored, the hacker performs its analysis in order to infer information about a secret data processed during the acquisition's period of time. This attack technique may be used to retrieve secret data such as keys. Side-channel analysis is usually performed by multiple executions of the same process in order to apply statistical analysis. Side-channel attacks have not proliferated in the handset hacking community yet because there are no secret keys in mobile equipment units. Nevertheless, this threat is growing with the increasing added value services integrated into handsets and smart-phones, as well as the rise of 3GPP networks. Indeed, we will soon witness the deployment of Digital Right Management [38] which specifies use of a DRM agent, content encryption keys and right encryption keys. It is in the interest of a handset malevolent owner to retrieve these keys in order to distribute protected content. It is obvious that handset hacking will increase at the same pace as benefits that can be obtained in return. Side-channel analysis is usually performed by the handset owner, but with contactless side-channel radiation it is possible to perform an attack on a nearby handset without the victim's knowledge. When keys are stored in handsets, a remote side-channel attack example is a hacker, physically close to his victims, retrieving authentication keys to bank accounts by means of a radiation sensor.

5.2.3 Fault Attacks

Fault attacks are another kind of hardware attack that emerged recently. This attack relies on a physical perturbation performed by the hacker rather then simply monitoring a side-channel. The core of the attack lies in the exploitation of the fault induced at the software level by the physical perturbation. There are many ways to perform a physical perturbation on an electronic device like a handset, the perturbation means being for example an electro-magnetic field, a power glitch or a laser beam. The exploitation technique is also variable and greatly depends on the target, which can be a cryptographic algorithm that may disclose secret information or an operating system sensitive process that might enable an unauthorized action such as a Midlet installation. Once again, the threat is real and will increase depending on the sensitivity of data stored in mobile equipment. As long as there are financial benefits in hacking a handset, the hacker will use any means to reach his goal. We refer the reader to [40] for a deeper treatment of fault attacks.

6 Conclusion

This chapter overviewed security features for the protection of mobile terminals and the attacks they are vulnerable to.

System architects should keep in mind that threats should be dealt with at the design level, the implementation level and the application use level. The previous sections provide examples of efforts made in multiple domains, their success and failure.

A typical security breach example at the design level occurred in the GSM authentication scheme. The lack of network authentication gave way to the possibility of setting up rogue base stations. Mutual authentication in 3GPP will eventually solve this problem. A careful implementation that follows scrupulously security guidelines will reduce the chance of faults at the implementation level. To mention a dangerous and widespread attack, lack of protection against buffer overflows can cause much damage, allowing for example access to protected memory areas. Application level attacks are probably the most prevalent. Mobile terminals are often accessed remotely, thereby greatly increasing the possibilities of runtime attacks. Moreover, users may exploit devices in a way they were not built for.

The large scale distribution of electronic devices and the increasing interaction among different technologies are not factors that will reduce security threats. Basic security rules apply to mobile terminals as to all other electronic devices. System security is that of its weakest link and the confidence in a system improves with the number of audits on it. Administrators should not rely on a single protection as attacks are multiple and on multiple levels.

References

[1] J. Muir, *Decoding Mobile Device Security*, 2003, `http://www.computerworld.com/mobile/mobiletopics/mobile`.

[2] Information Societies Technology (IST) Programme, *A Dependability Roadmap for the Information Society in Europe*, Project AMSD, Deliverable D 1.1, 2001.

[3] ANSI/IEEE Std 802.11, *Wireless LAN Medium Access Control (MAC) and Physical Layer Specifications (PHY)*, 1999 Edition.

[4] ANSI/IEEE Std 802.1x-2001, *Port-Based Network Access Control*, 2001 Edition.

[5] IEEE Std 802.11i/D3.0, *Draft Supplement to Standard for Telecommunications and Information Echange between Systems - LAN/MAN Specific Requirements. Specification for Robust Security*, February 2003 .

[6] Jovan Golic, Vittorio Bagini, Guglielmo Morgari, *Linear Cryptanalisys of Bluetooth Stream Cipher*, Springer-Verlag, LNCS 2332, pp. 238–255, EuroCrypt'02, 2002.

[7] Markus Jakobsson, Suzanne Wetzel, *Security Weaknesses in Bluetooth*, Springer-Verlag, LNCS 2020, RSA 2001, pp. 176–191, `http://www.rsasecurity.com/rsalabs/staff/bios/mjakobsson/bluetooth/bluetooth.pdf`.

[8] Scott Fluner, Itsik Mantin and Adi Shamir, *Weaknesses in the Key Scheduling Algorithm of RC4*, Selected Areas in Cryptography 2001.

[9] A. Stubblefield, J. Ioannidis, A. Rubin, *Using the Fluner Mantin and Shamir Attack to Break WEP*, AT&T Labs Technical Report TD-4ZCPZZ, August 6, 2001.

[10] Ron Rivest, *RSA security response to Weaknesses in the Key Scheduling Algorithm of RC4*, `http://www.rsasecurity.com/rsalabs/technotes/wep-fix.html`.

[11] Nikita Borisov, Ian Goldberg, David Wagner, *Intercepting Mobile Communications: The Insecurity of 802.11* , `http://www.isaac.cs.berkeley.edu/isaac/wep-draft.pdf`.

[12] William Arbaugh, Narendar Shankar and C. Justin Wan, *Your 802.11 wireless network has no clothes*, University of Marlyland, March 30, 2001.

[13] 3rd Generation Partnership Project, Technical Specification Group Services and System Aspects; 3G Security; Security Architecture (3G TS 33.102 version 6.0.0), September 2003.

[14] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Cryptographic Algorithm Requirements (3G TS 33.105 version 4.1.0), June 2001.

[15] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the 3GPP Confidentiality and Integrity Algorithms; Document 1: f8 and f9 Specification (3G TS 35.201 version 5.0.0), June 2002.

[16] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the 3GPP Personalization of Mobile Equipment; .

[17] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the MILENAGE Algorithm Set: An example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 2: Algorithm Specification (3G TS 35.201 version 5.1.0), June 2003.

[18] M. Walker. On the Security of 3GPP Networks. Invited Talk presented at EUROCRYPT 2000.

[19] Marc Briceno, Ian Goldberg and David Wagner. An Implementation of the GSM A3A8 Algorithm (Specifically COMP128), 1998, `http://www.iol.ie/~kooltek/a3a8.txt`.

[20] Paul Kocher, J. Jaffe and B. Jun. *Differential Power Analysis*, Springer-Verlag, LNCS 1666, pp. 388-397, Crypto'99, 1999.

[21] Jeremy Quirke. *Security in the GSM System*, May 2004, `http//www.ausmobile.com`.

[22] Josyula R. Rao, Pankaj Rohatgi, Helmut Scherzer and Stephane Tinguely. *Partitioning Attacks: Or How to Rapidly Clone Some GSM Cards*, 2002 IEEE Symposium on Security and Privacy, May 12-15, Berkeley, California, p.31, 2002.

[23] Klaus Vedder. *Security Aspects of Mobile Communications*, Computer Security and Industrial Cryptography, 1991.

[24] David Wagner. *GSM Cloning*, `http://www.isaac.cs.berkeley.edu/isaac/gsm.html`.

[25] A. Aho, R. Sethi, J. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, 1986.

[26] Z. Chen, *Java Card Technology for Smart Cards: Architecture and Programmer's Guide*, The Java Series, Addison-Wesley, 2000.

[27] R. Cohen, *The defensive Java virtual machine specification*, Technical Report, Computational Logic Inc., 1997.

[28] P. Cousot, R. Cousot, *Abstract Interpretation: a Unified Lattice Model for Static Analysis by Construction or Approximation of Fixpoints*, Proceedings of POPL'77, ACM Press, Los Angeles, California, pp. 238-252.

[29] K. Hypponen, D. Naccache, E. Trichina and A. Tchoulkine, *Trading-off type-inference memory complexity against communication*, Information and Communications Security (ICICS 2003), vol. 2836 of Lecture Notes in Computer Science, pp. 60-71, Springer-Verlag, 2003.

[30] X. Leroy, *Java Byte-Code Verification: an Overview*, In G. Berry, H. Comon, and A. Finkel, editors, Computer Aided Verification, CAV 2001, volume 2102 of Lecture Notes in Computer Science, pp. 265-285, Springer-Verlag, 2001.

[31] X. Leroy, *On-Card Byte-code Verification for Java card*, In I. Attali and T. Jensen, editors, Smart Card Programming and Security, proceedings E-Smart 2001, volume 2140 of Lecture Notes in Computer Science, pp. 150-164, Springer-Verlag, 2001.

[32] X. Leroy, *Bytecode Verification for Java smart card*, Software Practice & Experience, 32:319-340, 2002.

[33] T. Lindholm, F. Yellin, *The Java Virtual Machine Specification*, The Java Series, Addison-Wesley, 1999.

[34] N. Maltesson, D. Naccache, E. Trichina and C. Tymen, *Applet verification strategies for RAM-constrained devices*, Information Security and Cryptology – ICISC 2002, vol. 2587 of Lecture Notes in Computer Science, pp. 118-137, Springer-Verlag, 2003.

[35] G. McGraw, E. Felten *Securiy Java*, John Wiley & Sons, 1999.

[36] S. Muchnick, *Advanced Compiler Design and Implementation*, Morgan Kaufmann, 1997.

[37] G. Necula, *Proof-carrying code*, Proceedings of POPL'97, pp. 106-119, ACM Press, 1997.

[38] OMA DRM Specification 2.0.

[39] F-Secure Virus Descriptions : Cabir, `http://www.f-secure.com/v-descs/cabir.shtml`, June 2004.

[40] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall and C. Whelan, *The Sorcerers Apprentice Guide to Fault Attacks*, Cryptology ePrint Archive, Report 2004/100, 2004.