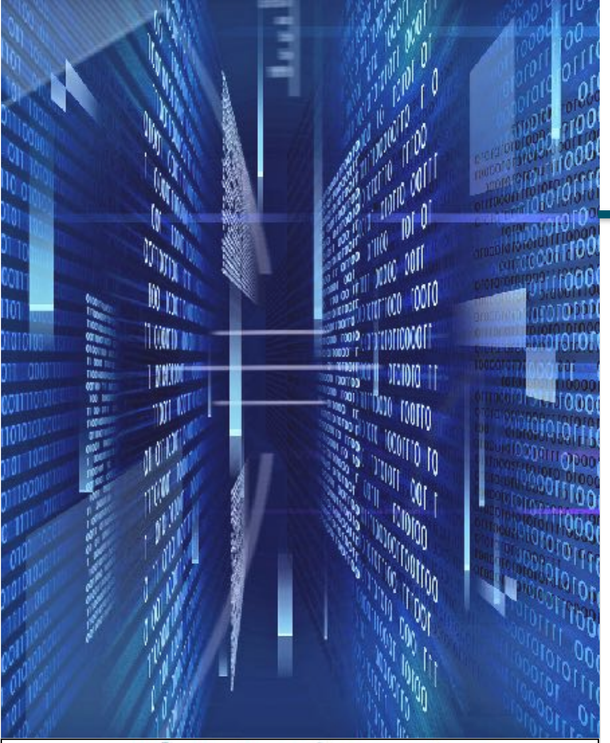

cybersecurity centre
GCHQ Academic Centre of Excellence EPSRC

UNIVERSITY OF Southampton

comp6224 (2016)
week 9: Telecoms & APIs

Vladimiro Sassone

Cyber Security Centre

University of Southampton

Cybersecurity Southampton

CyberSecuritySoton.org [w]

@CybSecSoton [fb & tw]

Telecoms

attacks on signalling

Until the 1980s, phone companies used signalling systems that worked *in-band* by sending tone pulses in the same circuit that carried the speech.

'*Phone phreaking*' refers to attacks on such signalling. Typically perpetrated via *blue boxes*.

Call an 0800 number and then send a 2600Hz tone that would *clear down* the line: disconnect the called party while leaving the caller with a trunk line connected to the exchange. One could now connect to a number without paying.

Phone phreaking was one of the roots of the computer hacker culture that took root in the Bay Area and was formative in the development and evolution of personal computers.

There was no way to stop blue-box type attacks so the phone companies spent years and many billions of dollars upgrading exchanges so that the signalling was moved out- of-band, in separate channels to which the subscribers had no easy access.

attacks on switching and configuration

The second wave of attacks targeted the computers that did the switching. Typically these were Unix machines on a LAN in the exchange. If you hacked into one, you could get quite some control.

A worrying recent development is the emergence of switching exploits by organisations.

The protocols used between phone companies to switch calls aren't particularly secure, as the move from in-band to out-of-band signalling was supposed to restrict access to trusted parties. But once again, changing environments undermine security assumptions.

mobile phones

The first generation of mobile phones used *analog signals* with no real authentication. The handset simply sent its serial numbers in clear over the air link. Villains built devices to capture these from calls in the neighbourhood. A black market developed in phone serial numbers.

It became a market for anonymous communications for criminals: enterprising engineers built mobile phones —known as *tumblers*— which used a different identity for each call: particularly hard to track.

Modern mobile phones are cellular, in that the operator divides the service area up into cells, each covered by a base station. The mobile uses whichever base station has the strongest signal, and there are protocols for handing off calls from one cell to another as the customer roams.

RF fingerprinting is a formerly classified military technology in which signal characteristics arising from manufacturing variability in the handset's radio transmitter are used to identify individual devices and tie them to the claimed serial numbers. This was suggested as a possible way to beat cloning and tumblers.

Another proposal was to adopt a *cryptographic authentication protocol*, but there are limits on how much can be done without changing the whole network.

For example, one can use a challenge-response protocol to modify the serial number. But many of the mechanisms people proposed to fortify the security of *analog* cellular phones have turned out to be weak.

Eventually they gave up and upgraded the whole thing to a new digital system:

- revenue protection
- digital systems use bandwidth more efficiently
- new features: international roaming, text messages, …

gsm security mechanism

The second generation of mobile phones adopted *digital* technology. Most handsets worldwide use the *Global System for Mobile Communications*, or GSM.

The designers of GSM set out to secure the system against cloning and other attacks. Goal: GSM should be at least as secure as the wireline system.

The industry initially tried to *keep secret* the cryptographic and other protection mechanisms at the core of GSM. *It didn't work*: some eventually leaked and the rest were reverse engineered.

Here we describe the basic authentication mechanisms.

gsm security mechanism

Each network has two databases to forward incoming calls to the correct cell:

- *home location register* (HLR): contains the location of the network's own mobiles
- *visitor location register* (VLR): location of mobiles roamed in from other networks.

The handsets are personalised using a *subscriber identity module* (SIM). These can be thought of as containing three numbers:

1. a *personal identification number* (PIN) to unlock the card to stop stolen mobiles being used. In practice, many networks set an initial PIN of 0000, and most users never change it or even use it;
2. there is an *international mobile subscriber identification* (IMSI), a unique number that maps on to your mobile phone number;
3. finally there is a *subscriber authentication key Ki*, a 128-bit number that serves to authenticate that IMSI and is *known* to your home network.

Keys are generated randomly and kept in an authentication database attached to the HLR.

gsm authentication mechanism

This is used to power the an authentication protocol as follows.

- on power-up, the SIM requests the customer's PIN, if this has been set;
- once the PIN is entered correctly, the SIM emits the IMSI to the handset;
- the handset sends the IMSI to the nearest base station;
- the base station relays the IMSI to the subscriber's HLR;
- the HLR generates five *triplets*, each consisting of:

  RAND, a random challenge (128bit); SRES, a response (32bit); and K_C, a ciphering key (64bit).

The relationship between these values is that RAND, encrypted under the SIM's authentication key *Ki*, gives an output which is SRES concatenated with K_C:

$$\{RAND\}_{Ki} = (SRES \,.\, K_C)$$

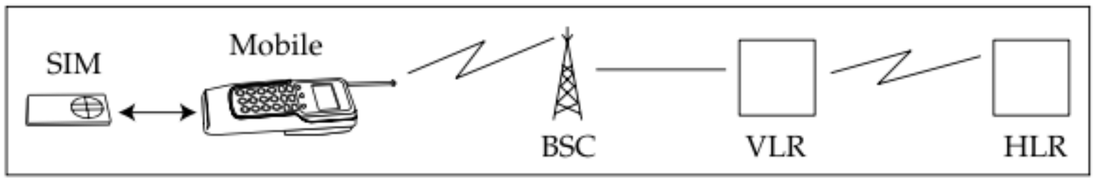

Figure 20.1: GSM authentication system components

gsm authentication protocol

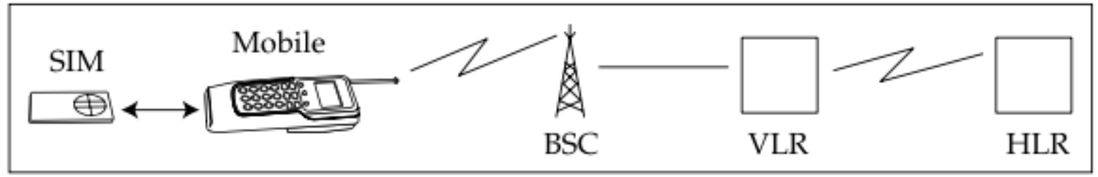

Figure 20.1: GSM authentication system components

$\{RAND\}_{Ki} = (SRES \,.\, K_C)$

The triplets are sent to the base station, which presents the first RAND to the mobile, which passes it to the SIM, which computes SRES. The mobile returns this to the base station and if it's correct the mobile and the base station can now communicate using the ciphering key K_C.

SIM → HLR	IMSI
HLR → BSC	(RAND, SRES, K_c), . . .
BSC → SIM	RAND
SIM → BSC	SRES
BSC → mobile	$\{traffic\}_{K_c}$

Figure 20.2: GSM authentication protocol

problems with the gsm protocol (1)

This encryption is done using algorithms called Comp128 (implementing A3/A8/A5).

Like most proprietary algorithms that were designed in the shadows and fielded quietly in the 1980s and 90s, it turns out to be vulnerable to cryptanalysis. That cracked the encryption.

This yet another example of the dangers of using a secret crypto primitive that has been evaluated by only a few friends; the cryptanalytic techniques necessary to find the flaw were well known and if Comp128 had been open to hostile public scrutiny, the flaw would most probably have been found.

Thankfully, a phone company can replace Comp128 with a proper hash function such as AES without affecting anyone else; the hash function is present only in the SIM cards it issues to its customers and the software at its HLR.

problems with the gsm protocol (2)

In most countries the communications between base stations and the VLR pass unencrypted on microwave links.

So an attacker could send out an IMSI of his choice and then intercept the triplet on the microwave link back to the local base station.

Also, triples can be replayed. An unscrupulous foreign network can get five triples while you are roaming on it and then keep on reusing them to allow you to phone as much as you want, even if the home network declines to. You might run high bills without knowing it.

Dishonest networks may also defraud roaming customers by *cramming* — by creating false billing records. So even if you thought you'd limited your liability by using a pre-paid SIM, you might still end up with your network trying to collect money from you.

location security

GSM is supposed to provide two further kinds of protection:

location security and call *content confidentiality*.

Location security: once a mobile is registered to a network, it is issued with a *temporary mobile subscriber identification* (TMSI).

This acts as its address as it roams through the network. The attack on this mechanism uses a device called an *IMSI-catcher*, which is sold to police forces.

This is based on proximity: being closer to the device than the base station, its signal is stronger and the mobile tries to register with it.

Then, the IMSI-catcher claims *not* to understand the TMSI, so the handset helpfully sends it the cleartext IMSI.

Notice that this feature cannot easily be disabled, as it is needed if mobiles are to be able to roam from one network to another without the call being dropped, and to recover from failures at the VLR.

content confidentiality

GSM provides call content confidentiality by encrypting the traffic between the handset and the base station once the authentication and registration are completed.

Speech is digitised, compressed and chopped into packets;

Each packet is encrypted by *xor*-ing it with a pseudorandom sequence generated from the ciphering key K_C and the packet number. The algorithm commonly used in Europe is A5/1.

An attack (found by Barkan, Biham and Keller) runs as follows:

1. record the call, including the initial protocol exchange of challenge and response;
2. once it is finished, you switch on your IMSI-catcher and cause the caller to register with your bogus base station;
3. the IMSI-catcher tells the mobile phone to use A5/2 rather than A5/1, and a key is duly set up: the IMSI-catcher sends the challenge that has captured before.

This vulnerability depends on the weaker A5/2 algorithm that was introduced following pressure from Europe's intelligence agencies, and the ostensible justification was that they didn't want even halfway decent security exported to potentially hostile countries, as it would be harder to spy on them.

summing up gsm

The initial GSM security mechanisms provided slightly better protection than the wireline network in countries allowed to use A5/1, and slightly worse protection elsewhere, until the protocol attacks were discovered and exploited.

Relatively few people ever get followed around by an investigator with professional snooping equipment — and if you're such a person, then ways to detect and prevent semi-active attacks on your mobile phone are just a small part of the tradecraft you need to know.

If you're an average subscriber, the privacy threat comes from possible abuse of data collected by the phone company, such as itemised billing data, more than from exploiting attacks carried out with professional snooping equipment.

3G

The third generation of mobiles was known as the *Universal Mobile Telecommunications System* (UMTS) and now as the *Third Generation Partnership Project* (3gpp, or just 3g).

Overall, the security is much the same as GSM, but upgraded to deal with a number of GSM's known vulnerabilities. The crypto algorithms A5/1, A5/2 and Comp128 are replaced by various a block cipher called *Kasumi*. All keys are now 128 bits.

Cryptography protects integrity and confidentiality of both content and signalling data (rather than just content confidentiality) and the protection at least runs from the handset to a fairly central node (rather than simply to the local base station). This means that picking up the triples, or the plaintext, from the microwave backhaul is no longer possible.

The authentication is now two-way, ending the vulnerability to rogue base stations: IMSI-catchers don't work against 3G mobiles.

There instead a properly engineered interface for *lawful interception*. It can supply keys as well as plaintext, so that if the police follow a suspect, record a call and identify the mobile in the process, they can decrypt that call later, rather than being limited to the plaintext of calls recorded by the phone company after they get their warrant approved.

3G authentication

The basic 3gpp protocol pushes authentication back from the BSC to the visitor location register. The HLR is known as the *home environment* (HE) and the SIM as the *UMTS SIM* (USIM).

The home environment chooses a random challenge RAND as before and enciphers it with the USIM authentication key *K* to generate a response RES, a confidentiality key *CK*, and integrity key *IK*, and an anonymity key *AK*.

$$\{\text{RAND}\}_K = (\text{RES} \,.\, \text{CK} \,.\, \text{IK} \,.\, \text{AK})$$

A sequence number SEQ is known to the HE and the USIM. A MAC is computed on RAND and SEQ. The sequence number is masked by xor'ing it with the anonymity key.

This is the *authentication vector AV* sent from the HE to the VLR.

The VLR then sends the USIM the challenge, the masked sequence number and the MAC;

The USIM computes the response and the keys, unmasks the sequence number, verifies the MAC, and if it's correct returns the response to the VLR.

USIM → HE	IMSI (this can optionally be encrypted)
HE → VLR	RAND, XRES, *CK*, *IK*, $SEQ \oplus AK$, *MAC*
VLR → USIM	RAND, $SEQ \oplus AK$, *MAC*
USIM → VLR	RES

Figure 20.4: 3gpp authentication protocol

3G protocols

The UMTS standards set out many other features, including details of sequence number generation, identity and location privacy mechanisms, backwards compatibility with GSM, mechanisms for public-key encryption of authentication vectors in transit from HEs to VLRs, and negotiation of various optional cryptographic mechanisms.

The net effect is that confidentiality will be improved over GSM: eavesdropping on the air link is prevented by higher-quality mechanisms, and the current attacks on the backbone network, or by bogus base stations, are excluded. Police wiretaps are done at the VLR.

summing up 3G

3G proposed to have end-to-end encryption, so that the call content and some of the associated signalling will be protected from one handset to another.

This led to government demands for a *key escrow protocol* to make keys available to police and intelligence services on demand.

Question is: if a mobile phone call takes place from a British phone company's subscriber using a U.S. handset, roaming in France, to a German company's subscriber roaming in Switzerland using a Finnish handset, and the call goes via a long distance service based in Canada and using Swedish exchange equipment, then which of these countries' intelligence agencies will have access to the keys?

The agencies in Britain and France implement the so-called Royal Holloway protocol, designed largely by Vodafone, which gives access to the countries where the subscribers are based (so in this case, Britain and Germany). This is achieved by using a variant of Diffie-Hellman key exchange in which the users' private keys are obtained by encrypting their names under a super-secret master key known to the local phone company and/or intelligence agency.

So 3G won't provide a revolution in confidentiality, merely an improvement. Its design goal is again that security should be comparable to wired network, and this looks like being achieved.

Security API attacks (mainly banking examples)

Simplicity is the ultimate sophistication.

— Leonardo Da Vinci

security API attacks

Secure devices typically have some kind of application programming interface, or API, that untrustworthy people and processes can call in order to get some critical task performed.

- A bank's server ask an hardware security module 'Here's a customer account number and PIN, with the PIN encrypted using the key we share with VISA. Is the PIN correct?'
- With javascript enabled, your browser exposes an application programming interface —javascript— which the owners of websites you visit can use to do various things.
- A secure operating system limits what an application program can make using e.g. a reference monitor to enforce policies such as preventing information flow from high to low.

This begs the question:

is it safe to separate tasks into a trusted component and a less trusted one?

It has recently been realised that the answer is very often no.

Designing security APIs is a very hard problem indeed.

ATM basics

Most ATMs operate using some variant of a system developed by IBM for its 3624 series cash machines in the late 1970s.

The card contains the customer's *primary account number*, *PAN*.

A secret key, called the '*PIN key*', is used to encrypt the account number. The decimalised and truncated result is called the '*natural PIN*'.

An offset can be added to it in order to give the PIN which the customer must enter. This has no real cryptographic function; it just enables customers to choose their own PIN.

Account number *PAN*:	8807012345691715
PIN key *KP*:	FEFEFEFEFEFEFEFE
Result of DES $\{PAN\}_{KP}$:	A2CE126C69AEC82D
$\{N\}_{KP}$ decimalized:	0224126269042823
Natural PIN:	0224
Offset:	6565
Customer PIN:	6789

Figure 10.3: IBM method for generating bank card PINs

offline ATMs

In the first ATMs to use PINs, each ATM contained:

a copy of the PIN key;

and each card contained:

the offset; the primary account number; withdrawal counter.

Each ATM could thus verify all customer PINs. Early ATMs also operated offline; if your cash withdrawal limit was $500 per week, a counter was kept on the card.

As networks became more dependable, ATMs have tended to operate online only. This simplifies the design: cash counters and offsets have vanished from magnetic strips and are kept on servers. Recently, magnetic strips have been supplemented with smartcard chips.

The basic principle remains: *PINs are generated and protected using cryptography.*

Dual control is implemented in this system using *tamper-resistant* hardware. A cryptographic processor, also known as a *security module*, is kept in the bank's server room and will perform a number of defined operations on customer PINs and on related keys in ways that enforce a dual-control policy.

In particular, the *dual control policy* will include:

1. Operations on the clear PINs and on the keys needed to compute and protect them, are all done in tamper-resistant hardware; the clear values are never made available to any bank's staff.
2. Cards and PINs are sent to the customer via separate channels. The cards are personalised in a facility with embossing and mag strip printing machinery, and the PIN mailers are printed in a separate facility containing a printer attached to a security module.
3. A *terminal master key* is supplied to each ATM in two printed components carried by two separate officials, input separately at the ATM keyboard, and automatically combined to form the key.
4. If ATMs are to perform PIN verification locally, then the PIN key is encrypted under the terminal master key and then sent to the ATM.
5. If the PIN verification is to be done centrally, the PIN is encrypted under a key set up using the terminal master key before being sent from the ATM to the security module for checking.
6. If the bank's ATMs are to accept other banks' cards, then its security modules use transactions that take a PIN encrypted under an ATM key, decrypt it and re-encrypt it for its destination, such as using a key shared with VISA. This *PIN translation* function is done entirely within the hardware security module, so that clear values of PINs are never available to the bank's programmers. VISA will similarly decrypt the PIN and re-encrypt it using a key shared with the bank that issued the card, so that the PIN can be verified by the security module that knows the relevant PIN key.

summing up on ATMs

During the 1980s and 1990s, hardware security modules became more and more complex.

The leading product is the IBM 4758.

But extending the dual control security policy from a single bank to tens of thousands of banks worldwide, as modern ATM networks do, was not completely straightforward.

back to API attacks

Multilevel secure systems impose flow controls on *static* data, while a security API is trying to prevent information flow in the presence of *tight* and *dynamic* interaction: much harder.

The most common API failure: transactions that are secure in isolation become insecure in combination. This can be because of application syntax, feature interaction, slow information leakage or concurrency problems.

A potentially critical example is '*Trusted Computing*'. This initiative has put a TPM chip for secure crypto key storage on most of the PC and Mac motherboards shipping today:

this allows applications to have a traditional '*insecure*' part running but also a '*secure*' part that will run on top of a new security kernel which will be formally verified and thus much more resistant to software attacks. These will guard crypto keys and other critical variables for applications.

The question this then raises is now the interface between the application and the NCA is to be protected.

Whenever a trusted computer talks to a less trusted one, the language they use is critical: expect that the less trusted device will try out all sorts of unexpected combinations of commands to trick the more trusted one. How can we analyse this systematically?

The XOR-To-Null-Key Attack (1)

‘*How can you be sure that there isn’t a chain of 17 transactions which will leak a clear key?*’

Hardware security modules are driven by transactions invoked by the host computers to which they are attached.

Each transaction typically consists of a command, a serial number, and several data items. The response contains the serial number and several other data items.

The security module contains a number of master keys that are kept in tamper-responding memory, which is zeroed if the device is opened.

However, there is often not enough storage in the device for all the keys it might have to use, so instead keys are stored encrypted outside the device. Furthermore the way keys are encrypted provides hints to their function

ATM security is based on dual control: need to generate two separate keys to be carried from the bank to the ATM, and entered into it at a keypad. The VISA device thus had a transaction to generate a key component and print out its clear value on an attached security printer. It also returned its value to the calling program, encrypted under the relevant master key *KM*, which was kept in the tamper-resistant hardware:

VSM → printer: *KMTi* ; VSM → host: $\{KMTi\}KM$

The XOR-To-Null-Key Attack (2)

It also had another transaction to combine two such components to produce a terminal key:

Host → VSM: $\{KMT1\}KM$, $\{KMT2\}KM$; VSM → host: $\{KMT1 \oplus KMT2\}KM$

To generate a terminal key for the first time, you'd use the first of these transactions twice followed by the second. Then you'd have $KMT = KMT1 \oplus KMT2$.

This allows programmer to take any old encrypted key and supplying it twice in the second transaction, resulting in a known terminal key (the key of all zeroes, as the key is xor'ed with itself):

Host → VSM: $\{KMT1\}KM$, $\{KMT1\}KM$; VSM → host: $\{KMT1 \oplus KMT1\}KM$

The module also had a transaction to encrypt any key supplied encrypted under *KM*, under any other key that itself is encrypted under *KM*. A programmer can retrieve the bank's PIN verification key — which is also kept outside the security module, encrypted with *KM*, and have it encrypted under the zero key that he now has inserted into the system:

Host → VSM: $\{0\}KM$, $\{PIN\}KM$; VSM → host: $\{PIN\}0$

The programmer can now decrypt the PIN verification key —the bank's crown jewels— and can work out the PIN for any customer account.

The key type 'communications key' is used for MAC keys, which have the property that you can input a MAC key in the clear and get it encrypted by 'any key that itself is encrypted under *KM*'. So you can put in an account number into the system, pretending it's a MAC key, and get it encrypted with the PIN verification key… Hmmm

The attack on the 4758

This particular hardware generates '*check values*' for keys: i.e., a key identifier calculated by encrypting a string of zeroes under the key. This is subject to a so-called *time-memory tradeoff* attack.

Suppose you want to crack a DES key of a type that you can't extract from the device. In a security module architecture it is generally enough to crack any key of a given type, as data can be converted back and forth between them. Also, one can generate large numbers of keys of any given type.

The attack therefore goes as follows.

1. Generate a large number of terminal master keys, and collect the *check value* of each.
2. Store all the check values in a hash table.
3. Perform a brute force search, by guessing a key and encrypting the fixed test pattern with it.
4. Compare the resulting check value against all the stored check values by looking it up in the hash table (an $O(1)$ operation).

With a 2^{56} key-space, an attacker who can generate 2^{16} target keys, a target key should be hit by luck with roughly $2^{56}/2^{16} = 2^{40}$ effort (which you can do in a week or so on your PC).

This is also called a '*meet-in-the-middle*' attack, reflecting the meeting of effort spent by generating keys and effort spent by the brute-force search checking keys.

PIN attack

One of the PIN block formats commonly used combines the PIN and the account number by exclusive-or and then encrypts them.

It turned out that exclusive-or was a bad choice: if the wrong account number is sent along with the PIN block, the device would decrypt the PIN block, xor in the account number, discover (with reasonable probability) that the result was not a decimal number, and return an error message.

The upshot was that by sending a series of transactions to the security module that had the wrong account number, you could quickly work out the PIN.

Account number *PAN*:	8807012345691715
PIN key *KP*:	FEFEFEFEFEFEFEFE
Result of DES $\{PAN\}_{KP}$:	A2CE126C69AEC82D
$\{N\}_{KP}$ decimalized:	0224126269042823
Natural PIN:	0224
Offset:	6565
Customer PIN:	6789

Figure 18.1: IBM method for generating bank card PINs

differential protocol analysis

As we know, the customer account number is encrypted using the PIN verification key, yielding a string of 16 hex digits.

The first four are converted to decimal digits for use as the PIN using a user-supplied function: the *decimalisation table*. This gives the natural PIN; an offset is then added whose function is to enable the customer to select a memorable PIN.

The problem is that the decimalisation table can be manipulated. If we set the table to all zeros (i.e., 0000000000000000) then a PIN of ’0000’ will be generated and returned in encrypted form.

We then repeat the call using the table 1000000000000000. If the encrypted result changes, we know that the DES output contained a 0 in its first four digits. Given a few dozen suitably- chosen queries, the value of the PIN can be found.

This neatly illustrates the difficulty of designing a device that will perform a secure multiparty computation, i.e., where a computation has to be performed using secret information from one party, and some inputs that can be manipulated by a hostile party.

The EMV Attack

The EMV consortium asked to add some transactions to the security module to support secure messaging between a *smartcard* and a bank *security module*.

The goal is that when a bank card appears in an online transaction, the bank can order it to change some parameter, such as a new key.

However, the specification had a serious error, which has appeared in a number of implementations

The transaction *'Secure Messaging For Keys'* allows the server to command the security module to encrypt a text message, followed by a key, with a key of a type for sharing with bank smartcards.

As the text message can be of variable length, an attacker can choose a message length so that just one byte of the target key crosses the boundary of an encryption block. That byte can then be determined by sending a series of messages that are one byte longer, where the extra byte cycles through all possible values until the key byte is found.

an example from operating systems

System call wrappers are used to beef up operating systems security; reference monitors are an example, and anti-virus software is another.

Wrappers intercept calls made by applications to the operating system, parse them, audit them, and may pass them on with some modification: for example, a *Low process* that attempts to access *High data* may get diverted to dummy data or given an error message.

Wrappers generally assume that system calls are atomic, but they're not; modern operating system kernels are very highly concurrent. This gives rise to *time-of-check-to-time-of-use* (TOCTTOU) attacks.

A typical attack is to race on user-process memory by getting the kernel to sleep at an inconvenient time, for example by a page fault. In an attack, a programme calls a path whose name spills over a page boundary by one byte, causing the kernel to sleep while the page is fetched; he then replaces the path in memory. It turns out that the race windows are large and reliable.

summary

There isn't much defence against API attacks: the only real solution is to rewrite the API.

In an ideal world, operating systems would move to a message-passing model, which would eliminate (or at least greatly reduce) concurrency issues. That's hardly practical for operating system vendors whose business models depend on backwards compatibility.

The APIs exposed by standard operating systems have a number of known weaknesses, but the wrapper solution, of interposing another API in front of the vulnerable API, looks extremely fragile in a highly concurrent environment.