6 Authentication

6.1 Introduction

Who are you? Depending on the situation you could answer this question by giving your name ("I'm Bob"), a group you belong to ("I'm a student") by giving the role you are playing (e.g. "I'm the bartender"), etc. Similarly your *identity* in a computer system can be a unique property of you (your user name, bank account number, public key) and/or something you share (being a student at the TUe) and/or something you are only part of the time (being a bartender in the weekend).

Exercise 3 *Validation that a user name indeed belongs to you is typically done with a password. How is this done for the other examples of identities given above?*

So you know who you are, but can you proof it? I.e. how can we validate your identity, how can you *authenticate* yourself? There are three basic categories of authentication methods:

- what you **have**; e.g. the key to your house, a bankcard, an rfid ov chip card, etc.
- what you **know**; e.g. the hiding place of the spare key, a security code (pincode), a password, your mothers maiden name, etc.
- what you **are**; e.g. your face which is recognized by your roommate, your voice on the phone, your behaviour (e.g. you're serving drinks), etc.

One can of course also combine different mechanisms, e.g. a bankcard with a security code combines the 'what you have' factor with the 'what you know' factor. Such multi-factor authentication is often used when there are high security requirements such as for use of your bankcard which represents a large monetary value; the *level of authentication* is matched with the requirements of the application. By using different factors attacks which target a single factor no longer work; e.g. pickpocketing to steal the 'what you have' (bankcard, a key) is not sufficient, nor is looking over your shoulder to learn what you know (security code, password). Both attacks have to be combined making it much harder to perform.

Exercise 4 *Recently credit cards have a chip embedded and require a security code to use. However, older cards also already used two factor authentication. Describe which factors are used and how for the old and new cards. Compare the two approaches and giving their relative advantages and disadvantages.*

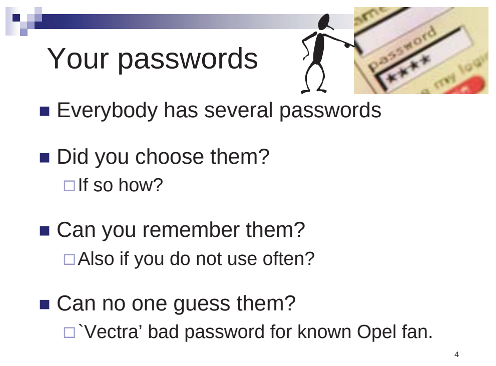
When you call a close friend you can typically suffice with an 'its me' as authentication; your friend will recognize you from you voice. Someone else, on the other hand, may only be able to recognize your voice after you've reminded them by stating your name.

We thus distinguish between *identification*, finding the identity of an individual (e.g. entering a user name), and *authentication*, proving that an individual indeed belongs to a (claimed) identity (e.g. entering a password). The processes of identification and authentication are not always as clearly separated as in the user name - password example.

Exercise 5 *For the examples above, consider which process/information is identification and which authentication. What are the advantages and disadvantages of combined identification-authentication?*

Below we consider mechanisms belonging to each of the three factors, beginning with passwords from the 'what you know' category.

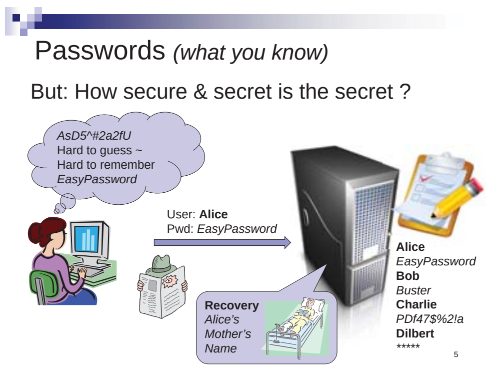
6.2 Passwords - What you know



Your passwords

- Everybody has several passwords
- Did you choose them?
  - If so how?
- Can you remember them?
  - Also if you do not use often?
- Can no one guess them?
  - `Vectra' bad password for known Opel fan.

4



Passwords *(what you know)*

But: How secure & secret is the secret ?

AsD5^#2a2fU
Hard to guess ~
Hard to remember
EasyPassword

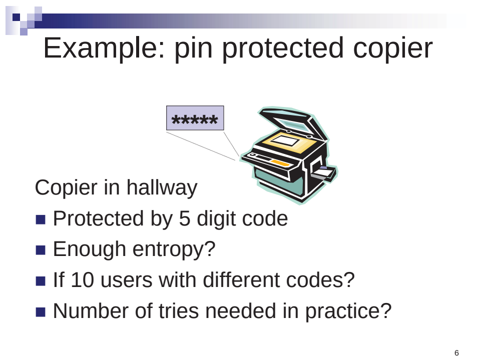
User: **Alice**
Pwd: *EasyPassword*

Recovery
Alice's Mother's Name

Alice
EasyPassword
Bob
Buster
Charlie
PDf47$%2!a
Dilbert
*****

5

Passwords are a very popular way of verifying your identity; they are familiar to the user and easy to implement on a system, e.g. requiring no new hardware or complicated programming. They are a key example of 'what you know' authentication. But do you actually know the password and are you the only one who does? I.e. can you remember your passwords and how strong is password authentication (e.g. can no one guess your password?)? A problem is that a hard to guess password will likely be hard to remember. The password also needs to be stored on the system and entered in the system when used to authenticate. Each opens up venues for attacks.

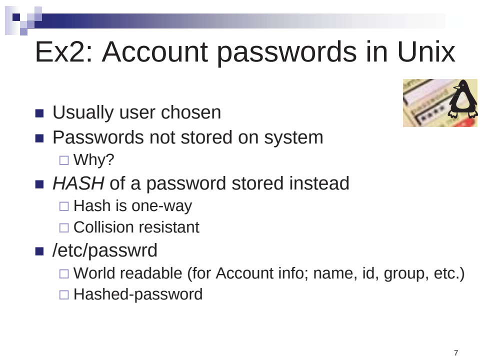


Example: pin protected copier

*****

Copier in hallway
- Protected by 5 digit code
- Enough entropy?
- If 10 users with different codes?
- Number of tries needed in practice?

6



Ex2: Account passwords in Unix

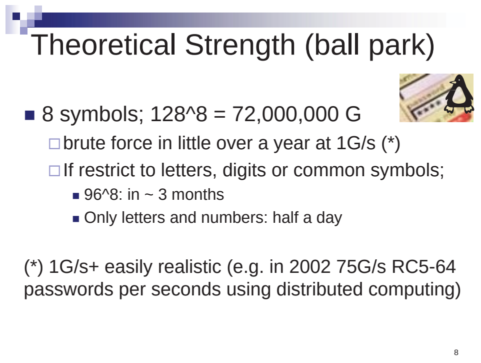
- Usually user chosen
- Passwords not stored on system
  - Why?
- *HASH* of a password stored instead
  - Hash is one-way
  - Collision resistant
- /etc/passwrd
  - World readable (for Account info; name, id, group, etc.)
  - Hashed-password

7

The amount of security offered by passwords is limited. Recall the discussion on strength of cryptographic operations; keys sizes such as 128 bites or even 2048 bits in public key settings were considered for cypher to offer practical security; with strength expressed in the bits of entropy provided e.g. '80-bits security'. So how many bits of entry does a pin code or a password have?

Consider a copier placed in a hallway that uses a 5 digit code to do both identification and authentication. There are 10^5 possible codes thus the entropy may seem to be log 2 of 100.000 and the average number of attempts needed to find the code thus 50.000. This could be considered to be enough; few attackers would try 50.000 codes (if they even get the chance) to be able to make free copies. However, we are doing combined identification and authentication. What if there are 10 users each with their own code. An attacker who is only interested in making free copies will now only have to find one of these codes. This reduces the work to an average of 5.000 tries. This may still seem to be reasonable. In reality the security was actually less. The actual number of tries we needed to break this system was .... one! So what is going on here?

Let's consider another example first; the password system of unix. Passwords on a unix system can usually be chosen by the user. They passwords are not stored on the system in the clear but rather in protected from. Newer implementations use hash functions such as MD5 to protect the password. The traditional approach used DES encryption. Note that it does not encrypt the password with some key; instead the password forms the key that is used to encrypt some fixed text (zero). (Which property of a cipher are we thus relying on?) A DES key is effectively 56 bits and a (printable) character can be expressed using 7 bits, giving that 8 characters will fill the key (additional characters are ignored). (Technically this is not hashing but below we will refer to passwords protected in this way also as 'hashed passwords'.)

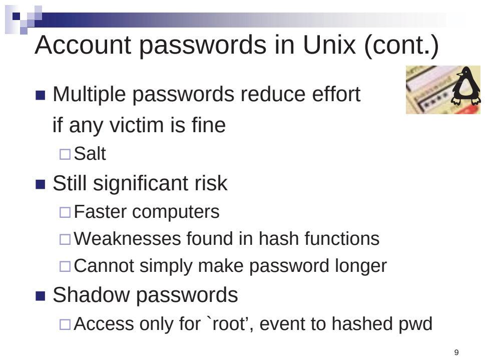


Theoretical Strength (ball park)

- 8 symbols; 128^8 = 72,000,000 G
  - brute force in little over a year at 1G/s (*)
  - If restrict to letters, digits or common symbols;
    - 96^8: in ~ 3 months
    - Only letters and numbers: half a day

(*) 1G/s+ easily realistic (e.g. in 2002 75G/s RC5-64 passwords per seconds using distributed computing)

8



Account passwords in Unix (cont.)

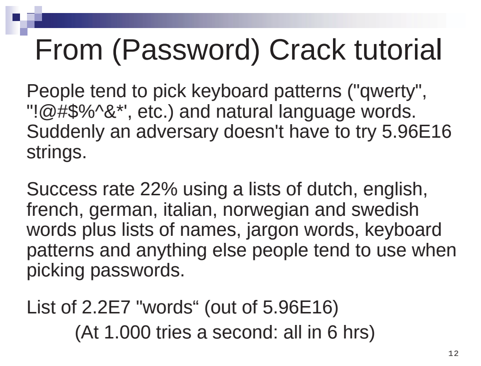
- Multiple passwords reduce effort if any victim is fine
  - Salt
- Still significant risk
  - Faster computers
  - Weaknesses found in hash functions
  - Cannot simply make password longer
- Shadow passwords
  - Access only for `root', event to hashed pwd

9

Even the weaker traditional scheme gives a theoretical 72 million billion possibilities. Of course billions of login attempts, if even feasible, will show that something is wrong. We can *limit the number of allowed failed login attempts* and even if set to a million the chance the attacker will succeed with random guessing is still less minute. However, what if the attacker manages to get at our database of protected passwords. Traditionally they are stored in a world readable file 'etcpasswd'. This file contains other account information as well that could be useful for some programs - and the passwords are hashed so no harm giving access to those right? *Right?!* (Newer systems move the hashed password out of this file to a 'shadow passwords' file which is readable only by 'root'; the most privileged user on Unix.)

To answer that question: no not right. With access to the hashed passwords an attacker can tryout different passwords without having to attempt to login; the only limit to the number of tries is the computational capabilities of the attacker. This makes breaking an 8 character password entirely feasible.

If the attacker does not care about which account is cracked, finding one of the passwords is sufficient. A salting scheme adds a random salt to the password before it is hashed. As different users will have different salt values, an attacker cannot compare a single hashed password guess against multiple users but will need to compute the hash for each user separately. (See also lab session.) Though this prevents additional weakening due to multiple users, the growing computational power of attackers create significant risks even then.



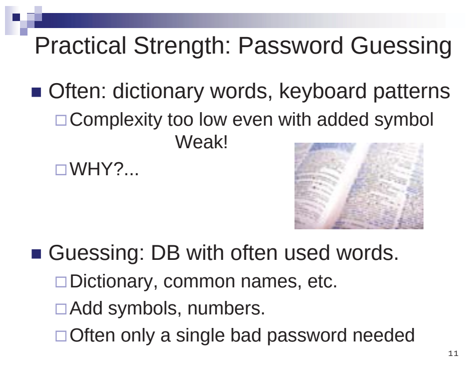
From (Password) Crack tutorial

People tend to pick keyboard patterns ("qwerty", "!@#$%^&*", etc.) and natural language words. Suddenly an adversary doesn't have to try 5.96E16 strings.

Success rate 22% using a lists of dutch, english, french, german, italian, norwegian and swedish words plus lists of names, jargon words, keyboard patterns and anything else people tend to use when picking passwords.

List of 2.2E7 "words" (out of 5.96E16)
(At 1.000 tries a second: all in 6 hrs)

12



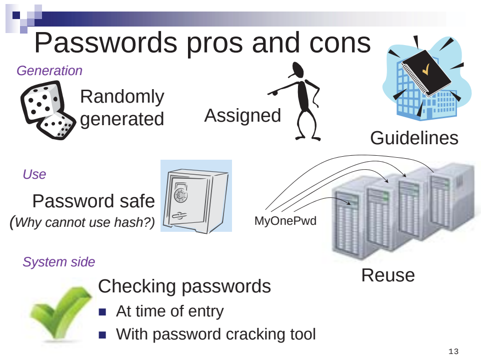
Practical Strength: Password Guessing

- Often: dictionary words, keyboard patterns
  - Complexity too low even with added symbol
    Weak!
  - WHY?...
- Guessing: DB with often used words.
  - Dictionary, common names, etc.
  - Add symbols, numbers.
  - Often only a single bad password needed

11

Random guessing attacks may be a problem for limited length passwords but attackers can do way better than that. Users typically do not choose their passwords by randomly generating a sequence. This may be a good way to get a password with high entropy, it will likely also result in a password that the user cannot easily remember. Instead users pick easy to remember patterns; such as names, words or keyboard patterns. A smart attacker will thus try those first significanly reducing the required effort.

Trying only several million common choices from the millions of billions of possibilities one can already find a good portion of the passwords.



Passwords pros and cons

Generation

Randomly generated

Assigned

Guidelines

Use

Password safe
(*Why cannot use hash?*)

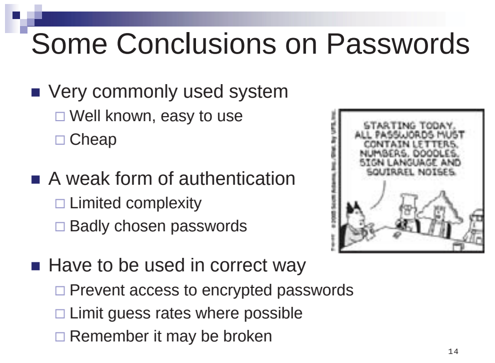
MyOnePwd

Reuse

System side

Checking passwords
- At time of entry
- With password cracking tool

13



Some Conclusions on Passwords

- Very commonly used system
  - Well known, easy to use
  - Cheap
- A weak form of authentication
  - Limited complexity
  - Badly chosen passwords
- Have to be used in correct way
  - Prevent access to encrypted passwords
  - Limit guess rates where possible
  - Remember it may be broken

STARTING TODAY, ALL PASSWORDS MUST CONTAIN LETTERS, NUMBERS, DOODLES, SIGN LANGUAGE AND SQUIRREL NOISES.

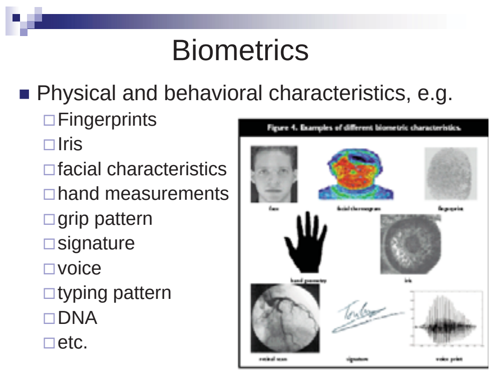
14

Different techniques to combat weak passwords each have their own advantages and disadvantages. Randomly generated passwords are stronger but harder to remember. Guidelines on how to construct 'good' passwords should not be too specific as that also gives information to an attacker, i.e. decreases the entropy. Reusing the same password on multiple systems only gives one that needs to be remembered (so perhaps it can be stronger) but means that a break-in in one system (or a malicious administrator) will affect the security of other systems as well. A password safe can be used to store different strong (e.g. randomly generated) passwords on different systems but does create a single point of failure, both with respect to confidentiality (all passwords protected by a single master password) and availability (if safe unavailable or master password forgotten then all passwords are lost).

Overall passwords form an easy to use but not very strong form of authentication. Protecting high value resources with only a password is not advisable.

6.3 Biometrics - What you are

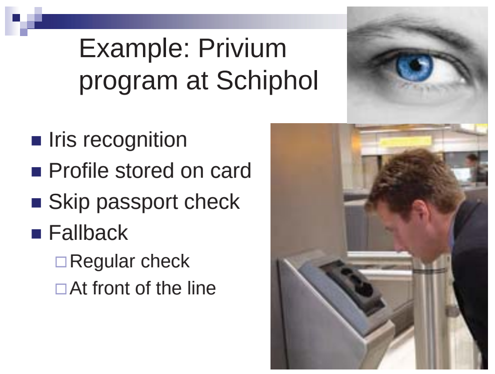
When you meet someone you know you will recognize their face, recognize the way they walk, etc. When someone brings you a menu you identify them as the waiter. You may identify someone hanging around a shop carrying bags and watching the sales person instead of shopping as a potential shoplifter. When immigration compares you to the picture on your passport. Biometrics are used; physical and/or behavioural aspects of the subject. Clearly these fall under the 'what you are' category.



Biometrics

- Physical and behavioral characteristics, e.g.
  - Fingerprints
  - Iris
  - facial characteristics
  - hand measurements
  - grip pattern
  - signature
  - voice
  - typing pattern
  - DNA
  - etc.

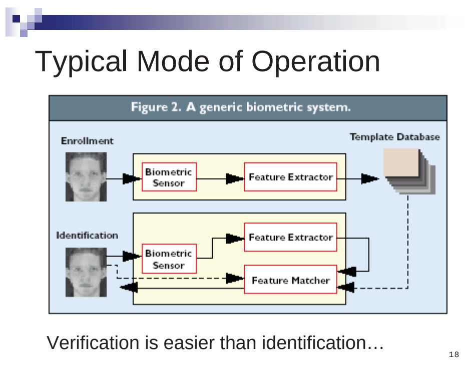
Figure 4. Examples of different biometric characteristics.



Example: Privium program at Schiphol

- Iris recognition
- Profile stored on card
- Skip passport check
- Fallback
  - Regular check
  - At front of the line

Examples of applications of biometrics include the privium program at Amsterdam Schiphol airport which enables frequent flyers to be authenticated by means of iris recognition so they can avoid the (lines at the) passport check. Some laptops include a fingerprint scanner which can be used instead of a password. Passports contain two different biometrics; pictures and fingerprint information.

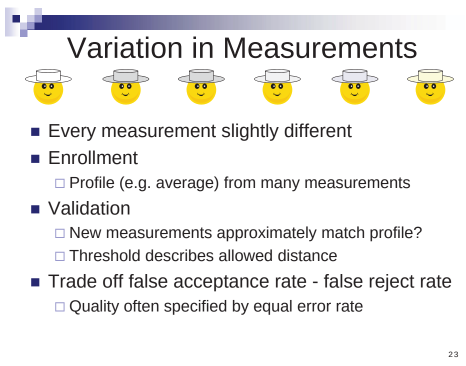


Typical Mode of Operation

Figure 2. A generic biometric system.
Enrollment
Template Database
Biometric Sensor
Feature Extractor
Identification
Biometric Sensor
Feature Extractor
Feature Matcher

Verification is easier than identification…

18



Variation in Measurements

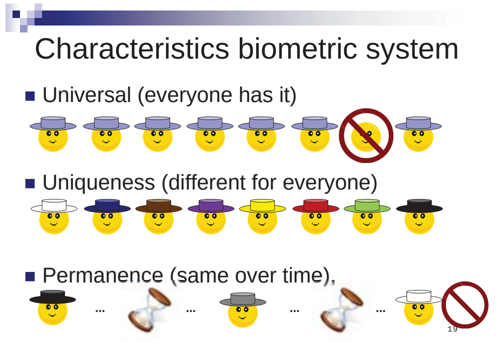
- Every measurement slightly different
- Enrollment
  - Profile (e.g. average) from many measurements
- Validation
  - New measurements approximately match profile?
  - Threshold describes allowed distance
- Trade off false acceptance rate - false reject rate
  - Quality often specified by equal error rate

23

A biometric based identification system typically operates in two phases; first there is the *enrollment* phase in which a *template* is created for each subject. The template consists of scores on multiple *features* which are quantifiable aspects (e.g. the distance between the eyes, length of the nose, distance nose-mouth, etc.) extracted from a measurement done by some *biometric sensor* (e.g. a camera, fingerprint scanner, etc.).

There are always variations in the biometric measurements; noise, imperfect extraction, but also changes over time; the biometric of the subject is not exactly the same as when the enrolled happened. For enrolment this means we will typically want to use multiple measurements to help reduce noise and ensure the profile reflects the 'typical' measurement for the user. For validation it means that we will need to allow for small variations in the measurement. But what is small; i.e. what is the threshold for allowed variation ? Here we have a trade-off between *false acceptance* and *false rejection*. If we set a high threshold (i.e. allow larger variation) this means it is unlikely that we will reject the correct person i.e. a low false reject rate (also called 'insult rate'). However, it also gives a higher false acceptance rate (also called 'fraud rate'); the chance of an incorrect match is increased. Setting a low threshold reverses this giving a high false reject but low false accept rate. There is no universal 'right setting' for the threshold; one has to choose this depending on the application. To still have a single number that can be used to compare two biometric schemes one often looks at the equal error rate - i.e. the threshold is chosen such that the false accept and false reject rate are the same. Obviously the lower this equal error rate is the better.

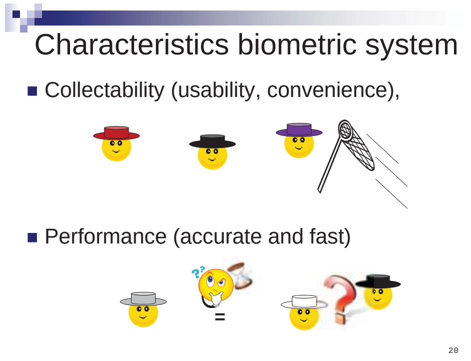
With passwords we have seen that identification is harder than authentication; we loose entropy if we use the password to both identify and authenticate instead of only authenticate the user. We have a similar situation with biometrics; validating an identity claim requires comparing a biometric measurement against a single profile while identification requires comparing against each entry in the database, increasing the change for an incorrect match. Note that the privium example does biometric authentication; a card provides the claimed identity (as well as storage for the profile).



Characteristics biometric system

- Universal (everyone has it)
- Uniqueness (different for everyone)
- Permanence (same over time),

19



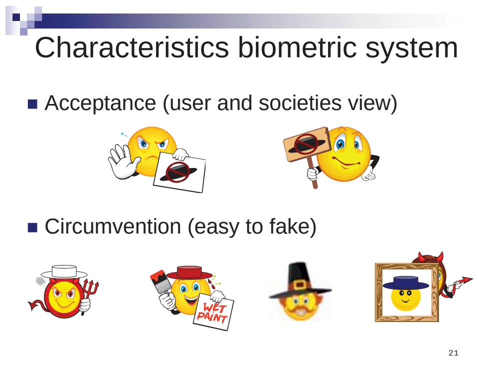
Characteristics biometric system

- Collectability (usability, convenience),
- Performance (accurate and fast)

20

A good biometric has the following characteristics:

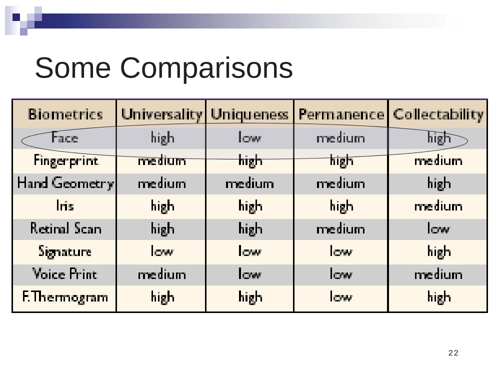
- *Universality*; everyone (within the target audience) has it. Voice recognition is not possible for a mute person, signature not for someone who cannot write, fingerprints not for people who have done a lot of manual labor with their hands as fingerprints can 'ware off', etc.
- *Uniqueness*; it is different for everyone. The more unique a profile, the stronger the authentication provided by a match.
- *Permanence*; it stays the same over time. If the biometric changes between enrollment and use then the profile will no longer match the new measurement.
- *Collectability*; usability, convenience. Useability of the system depends on how easy it is to take a biometric measurement (e.g. do I have to stand straight in front of the camera, at a fixed distance, do I have to type a page of text (to recognize typing pattern) everytime want to log on ...)
- *Performance*; accurate and fast. It has to be possible to measure and extract features consistently and within a reasonable amount of time/effort.
- *Acceptance*; user and societies view. Users may not accept that their finger print/picture/... is collected (for the purpose of authenticating to the system).
- *No Circumvention*; not easy to fake. Fake fingerprint - photo in front of camera - etc. Note that this also dependent on the way the biometric system is used, how the measurement taken; e.g. could I hold a photo in front of the camera or is there a supervisor that checks that I am standing there.



Characteristics biometric system

- Acceptance (user and societies view)
- Circumvention (easy to fake)

21

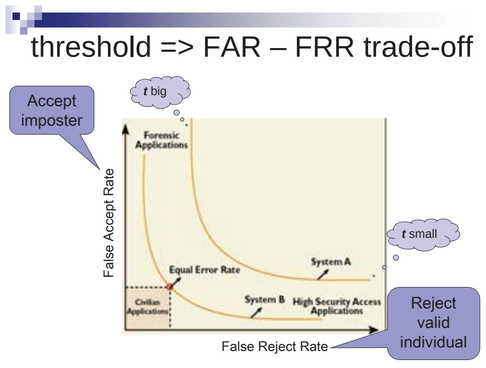


Some Comparisons

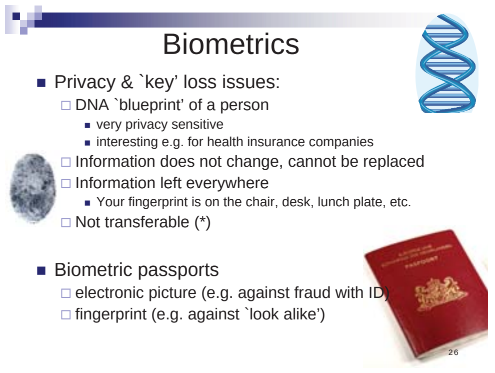
Biometrics	Universality	Uniqueness	Permanence	Collectability
Face	high	low	medium	high
Fingerprint	medium	high	high	medium
Hand Geometry	medium	medium	medium	high
Iris	high	high	high	medium
Retinal Scan	high	high	medium	low
Signature	low	low	low	high
Voice Print	medium	low	low	medium
F.Thermogram	high	high	low	high

22

Choosing a biometric suitable for your application is a matter of the trade-offs between different characteristics. If ease of use is important then high collectability is needed, for strong authentication a high uniqueness is required. Along with the biometric one also should consider what threshold to choose, making a trade-off between false acceptance and false rejection rates.



threshold => FAR – FRR trade-off



Biometrics

- Privacy & `key' loss issues:
  - DNA `blueprint' of a person
    - very privacy sensitive
    - interesting e.g. for health insurance companies
  - Information does not change, cannot be replaced
  - Information left everywhere
    - Your fingerprint is on the chair, desk, lunch plate, etc.
  - Not transferable (*)
- Biometric passports
  - electronic picture (e.g. against fraud with ID)
  - fingerprint (e.g. against `look alike')

26

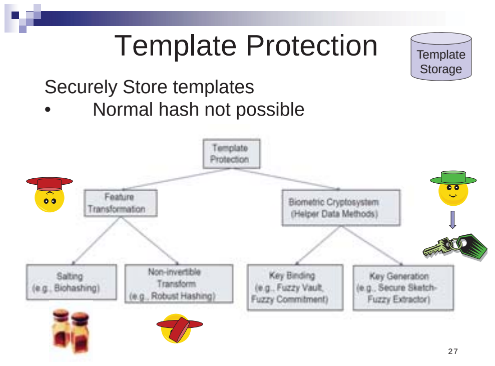
The type of application and how well we can deal with false accepts/rejects influence the threshold that we choose; In forensics application one is looking for suspects to investigate further and false rejects (missed the culprit) are far worse than false accepts (additional suspects to look at): a high threshold is chosen. In high security applications false accepts are far worse than false accepts which likely can be dealt with in another way (e.g. like in privium example where after a failed authentication one can revert to the normal passport check (without having to wait in line)): a low threshold is chosen. In many civilian applications both false rejects and false accepts are similarly undesirable and allowing one to be high to improve the other a little bit is not acceptable; a medium threshold is chosen.

You cannot transfer your biometric. What you know or have you can give to someone else but not so with what you are. This may seem like a good property for identification - however, typically the end goal is to authenticate subjects and assign them some rights (this is the owner of the car and she may start the car) rather then identify them (this is Alice). We have to be careful that we are achieving the right goal and that we do not threaten a more important goal (e.g. keeping your finger...).

When someone finds your password, you choose a new one. If you lose your key you replace the lock. A biometric is not replaceable; you cannot change your fingerprints. A biometric does not remain secret; you leave your fingerprint on the glass you use, everyone around you can see your face (which may be relevant with respect to circumvention). Despite not being very secret they can be very they can still very privacy sensitive (e.g. Your DNA, pictures of you, etc.). Combined this means that storing a template unprotected is not a good idea. (Note that the database should also be protected against insiders e.g. a database admin. Encrypting the profiles provides no protection against someone that can get the decryption key.)

Template Protection With passwords we used a hash as a one-way transformation. By storing the hash the passwords are not (immediately) recoverable even if we manage to get access to the system. To check that an entered password is correct we hash it again and compare it to the stored hash value. We cannot use a hash in this way with biometric templates; a new biometric measurement should be close to the stored template, but will not be exactly the same. As such its hash will be completely different from the hash of the stored template. (Recall that a good hash is a random function; two inputs that are close together will still result in completely different outputs.) Other protection schemes are needed.

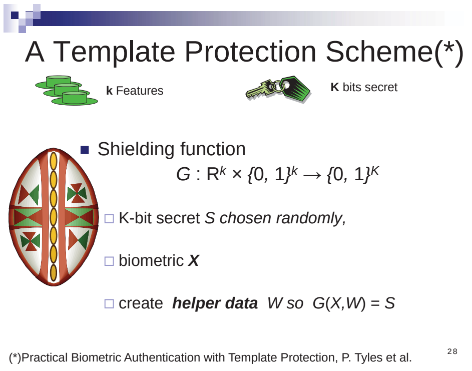
One class of template protection schemes apply transformations to the features; trying to make the stored feature reveal little about the biometric itself (e.g. a picture that is warped so it is no longer be recognizable). Adding noise or applying one-way transformation that do map close points to closely related points. Another class tries to extract a binary key from the biometric. Below we look at the setup of one such a system.



Template Protection

Securely Store templates
- Normal hash not possible

Template Storage
Template Protection
Feature Transformation
Biometric Cryptosystem (Helper Data Methods)
Salting (e.g. Biohashing)
Non-invertible Transform (e.g. Robust Hashing)
Key Binding (e.g. Fuzzy Vault, Fuzzy Commitment)
Key Generation (e.g. Secure Sketch-Fuzzy Extractor)
27



A Template Protection Scheme(*)

k Features — **K** bits secret

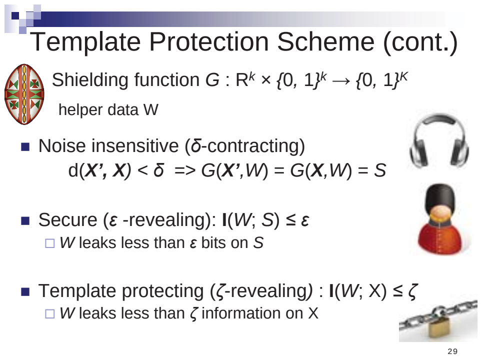
- Shielding function

  $G : R^k \times \{0, 1\}^k \to \{0, 1\}^K$

  - K-bit secret *S chosen randomly,*
  - biometric ***X***
  - create ***helper data*** *W so G(X,W) = S*

(*)Practical Biometric Authentication with Template Protection, P. Tyles et al.

To translate a biometric measurement into a binary key, a shielding function is used. First a K-bit key is randomly generated for the user. The shielding function will map the biometric template to the K-bit of this user. To be able to do this the function uses k bits of so called helper data. Instead of the template we now store the helper data. For a new biometric measurement we apply the shielding function to the new measurement and the stored helper data to obtain the key (which can then be verified).



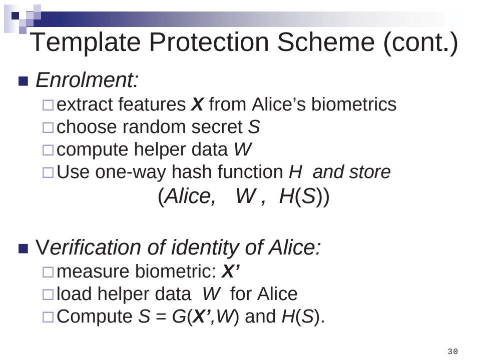
Template Protection Scheme (cont.)

Shielding function $G : \mathrm{R}^k \times \{0,1\}^k \rightarrow \{0,1\}^K$

helper data W

- Noise insensitive (δ-contracting)
  $d(\boldsymbol{X'}, \boldsymbol{X}) < \delta \Rightarrow G(\boldsymbol{X'},W) = G(\boldsymbol{X},W) = S$
- Secure (ε-revealing): $\mathbf{I}(W; S) \leq \varepsilon$
  - W leaks less than ε bits on S
- Template protecting (ζ-revealing) : $\mathbf{I}(W; X) \leq \zeta$
  - W leaks less than ζ information on X

29



Template Protection Scheme (cont.)

- *Enrolment:*
  - extract features $\boldsymbol{X}$ from Alice's biometrics
  - choose random secret S
  - compute helper data W
  - Use one-way hash function H *and store*
    $(Alice, \quad W, \quad H(S))$
- *Verification of identity of Alice:*
  - measure biometric: $\boldsymbol{X'}$
  - load helper data W for Alice
  - Compute $S = G(\boldsymbol{X'},W)$ and $H(S)$.

30

The shielding function should obviously also make sure that measurements close to the template map to the same key so the correct key is recovered. It should also make sure that the key remains secure; the helper data should not reveal the key. Finally, as our goal is to protect the template, the helper data should not reveal the template.

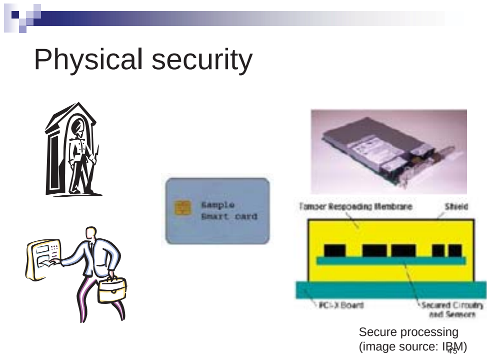
Biometrics conclusions:
-Varying strength of identification
-Can be tailored to the application
-Additional hardware needed
-Non-replaceable
-Privacy & Acceptance

6.4 Hardware tokes - What you have

Example Tokens

Functional & Security Goals

44

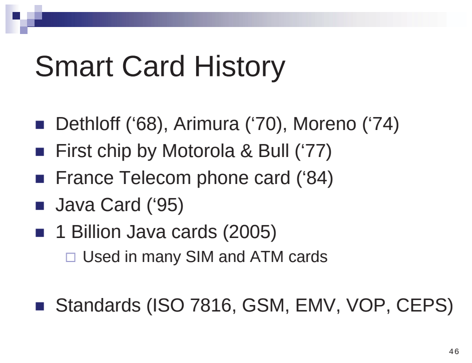


Physical security

Secure processing
(image source: IBM)

45

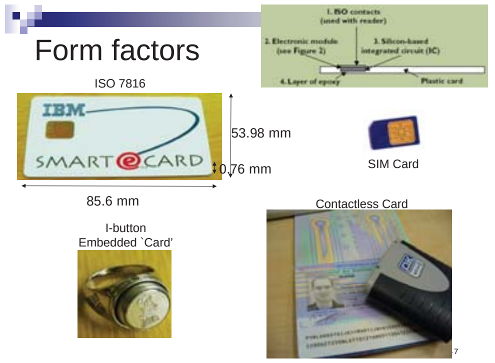
Hardware security tokens, e.g. a key to your house, a credit card, your ov chip card, your telephone, fall in the 'what you have' category. (Note that when we look at how we can be sure that 'what you have' is actually the what we are looking, i.e. when we try to authenticate your token, we again see the same factors appearing; we can use e.g. its shape (what it is) for your house key or information that it stores (what it 'knows') for your credit card, or a sim card (what it has) for your phone.)



Smart Card History

- Dethloff ('68), Arimura ('70), Moreno ('74)
- First chip by Motorola & Bull ('77)
- France Telecom phone card ('84)
- Java Card ('95)
- 1 Billion Java cards (2005)
  - Used in many SIM and ATM cards
- Standards (ISO 7816, GSM, EMV, VOP, CEPS)

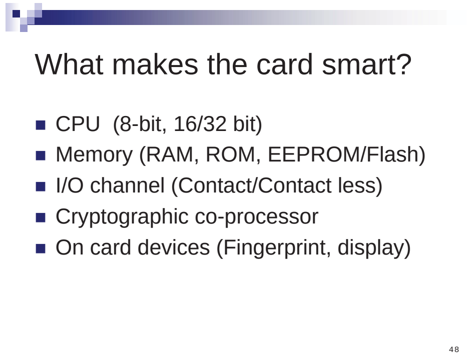
46



Form factors

7

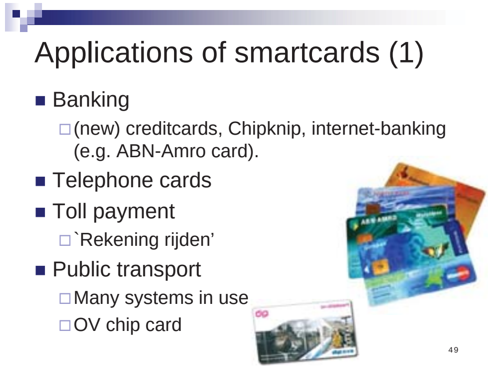
Physical tokens may contain secrets that need to be protected; in case of loss of the token but possibly also against the holder of the token herself. Consider the example of the chipknip where the card stores its own balance; the holder should not be able to raise the balance without paying for it.



What makes the card smart?

- CPU (8-bit, 16/32 bit)
- Memory (RAM, ROM, EEPROM/Flash)
- I/O channel (Contact/Contact less)
- Cryptographic co-processor
- On card devices (Fingerprint, display)

48

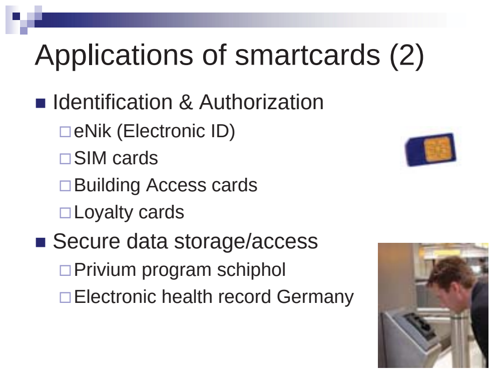


Applications of smartcards (1)

- Banking
  - (new) creditcards, Chipknip, internet-banking (e.g. ABN-Amro card).
- Telephone cards
- Toll payment
  - `Rekening rijden'
- Public transport
  - Many systems in use
  - OV chip card

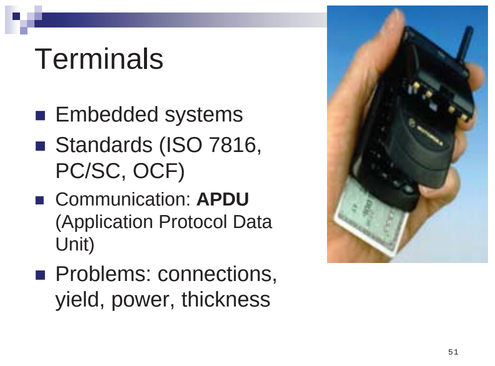
49

As the holder has access to the token, physical tampering with the hardware to extract or change information on the token is a threat that needs to be addressed; the token has to be made tamper resistant. Smart cards are well known examples of tamper resistant devices with many, sometimes high value, applications. As such they are interesting targets for attackers.



Applications of smartcards (2)

- Identification & Authorization
  - eNik (Electronic ID)
  - SIM cards
  - Building Access cards
  - Loyalty cards
- Secure data storage/access
  - Privium program schiphol
  - Electronic health record Germany

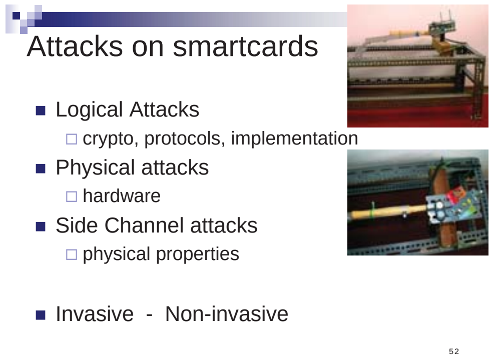


Terminals

- Embedded systems
- Standards (ISO 7816, PC/SC, OCF)
- Communication: **APDU** (Application Protocol Data Unit)
- Problems: connections, yield, power, thickness

51

Logical attacks against smartcards use flaws in the design of the system; the use of weak crypto or protocols, mistakes in the implementation. Because the attacker has physical access to the device, however, there are additional attack possibilities; side channel attacks in which the attacker learn more by observing the physical properties of the device and hardware attacks where the hardware is manipulated to e.g. extract a secret.

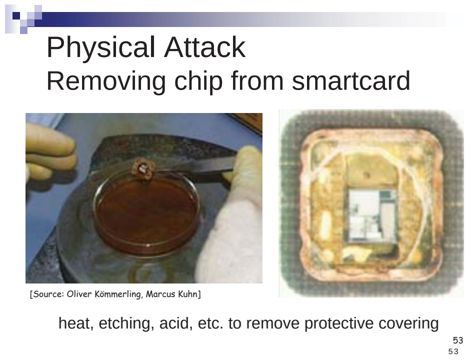


Attacks on smartcards

- Logical Attacks
  - crypto, protocols, implementation
- Physical attacks
  - hardware
- Side Channel attacks
  - physical properties

- Invasive - Non-invasive

52



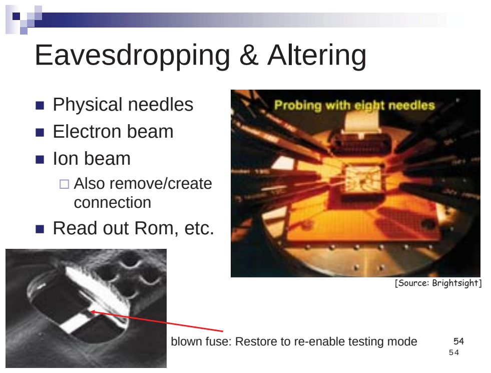
Physical Attack
Removing chip from smartcard

[Source: Oliver Kömmerling, Marcus Kuhn]

heat, etching, acid, etc. to remove protective covering

53
53

Reading out the rom memory, either directly or by re-enabling a testing mode that was disabled before issuing the card, breaking or creating connections using an ion beam and eavesdropping channels on the chip by physically connecting to them are examples of hardware attacks that are employed in smartcard testing labs.



Eavesdropping & Altering

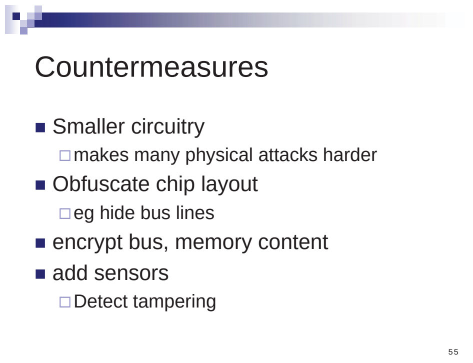
- Physical needles
- Electron beam
- Ion beam
  - Also remove/create connection
- Read out Rom, etc.

Probing with eight needles

[Source: Brightsight]

blown fuse: Restore to re-enable testing mode

54
54

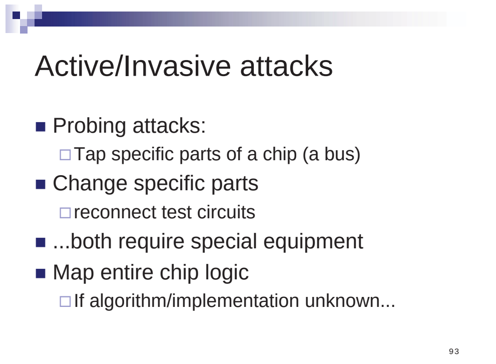


Countermeasures

- Smaller circuitry
  - makes many physical attacks harder
- Obfuscate chip layout
  - eg hide bus lines
- encrypt bus, memory content
- add sensors
  - Detect tampering

55

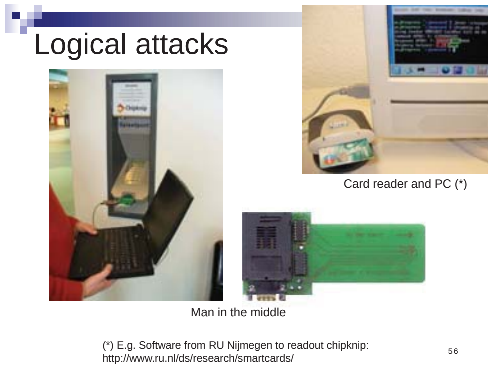
Several logical attacks have made the news; the ov-chipcard being a key example. These attacks are not specific to hardware tokens and can use flaws in the design (see also the first lecture, Chapter **??**), the access control (see Chapter **??**), the protocols (see next lecture; Chapter **??**), etc.



Active/Invasive attacks

- Probing attacks:
  - Tap specific parts of a chip (a bus)
- Change specific parts
  - reconnect test circuits
- ...both require special equipment
- Map entire chip logic
  - If algorithm/implementation unknown...

93



Logical attacks

Card reader and PC (*)

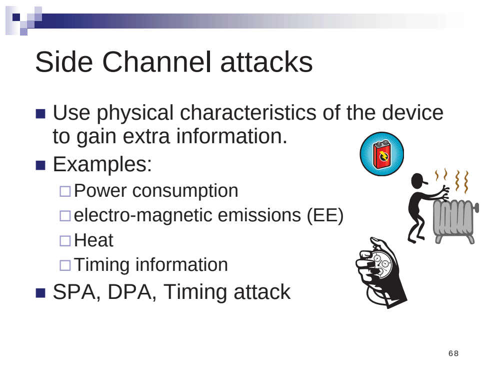
Man in the middle

(*) E.g. Software from RU Nijmegen to readout chipknip: http://www.ru.nl/ds/research/smartcards/

56

6.4.1 Side Channel Attacks

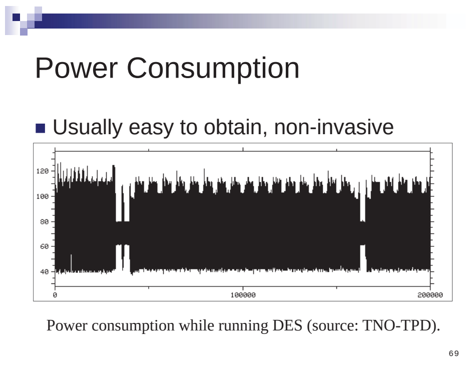
When analysing a program, such as an encryption function, we typically look at its input-output behaviour; e.g. does the key remain hidden if the attack can see the plain text (input channel) and cipher text (output channel). Side channel attacks use physical characteristics of the device as an additional channel of information. For example by looking how long a computation takes we may learn about how 'difficult' it is, which in turn may tell us something about the secrets used. How much power a device consumes while running the program, any electro-magnetic emissions, and heat it produces are examples of potential side channels.



Side Channel attacks

- Use physical characteristics of the device to gain extra information.
- Examples:
  - Power consumption
  - electro-magnetic emissions (EE)
  - Heat
  - Timing information
- SPA, DPA, Timing attack

68



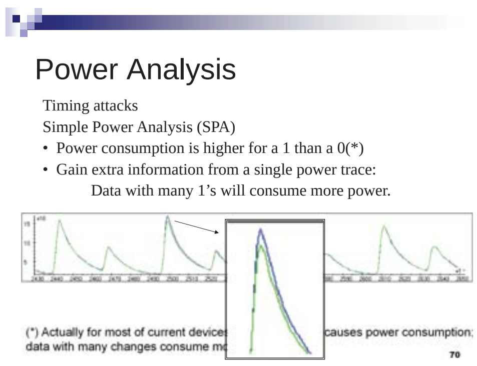
Power Consumption

- Usually easy to obtain, non-invasive

Power consumption while running DES (source: TNO-TPD).

69

If we measure the power consumption of a device running DES (see Section **??** on DES) we can easily recognize the phases of the computation; preprocessing is followed by sixteen similar rounds (the Feistel rounds) and then some post processing. We can recognize the cipher being used, which may already be of some use. However, if we look at the power consumption in detail we can do much more than this.

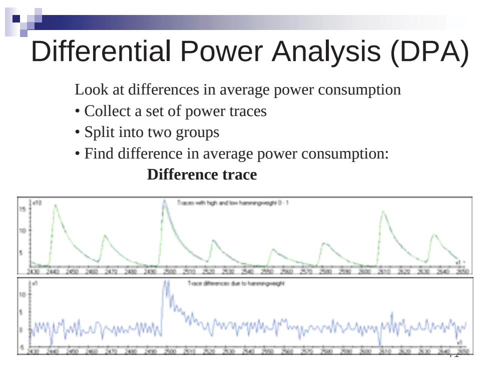


Power Analysis

Timing attacks
Simple Power Analysis (SPA)
- Power consumption is higher for a 1 than a 0(*)
- Gain extra information from a single power trace:
  Data with many 1's will consume more power.

(*) Actually for most of current device … causes power consumption;
data with many changes consume m…

70



Differential Power Analysis (DPA)

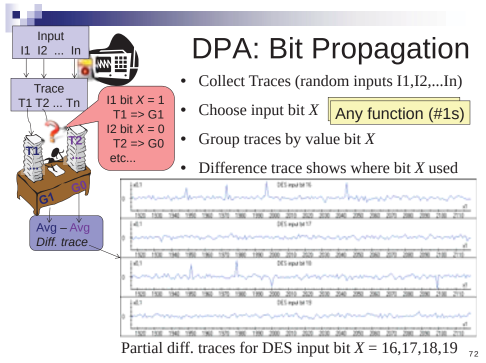
Look at differences in average power consumption
- Collect a set of power traces
- Split into two groups
- Find difference in average power consumption:
  Difference trace

The power consumption allows us to see what phase of the computation starts when. This strengthens our timing attacks; instead of only having the overall time it takes the algorithm to complete we know how long each phase takes. Below we will see an example for RSA where this is already enough to easily extract the key.

Looking not only at the general shape of the power consumption trace but also at the heights of the peaks, we can try to extract information directly. If at time stamp 2500 the algorithm is using a 1, it will consume more power then when it is working with a 0. (Actually changes in values being put on the internal memory bus of the device are what causes much of the power consumption but this detail is not important for the analysis. Often 'many 1s equals high consumption' is a valid assumption. For devices where this is not the case the type of power analysis discussed here may still be possible by adjusting this assumption, i.e. by using a different 'power model'.)

If we know when a given bit of the key is used we can perform a Simple Power Analysis (SPA) attack in which we examine the power trace at that location and judge whether the consumption at that point is 'high' or 'low' to determine whether that bit is one or zero. Difficulties are finding the exact location in the trace, the small difference between 'low' and 'high'; it will be hard to know whether a score is 'high' without having something to compare it with and noise in the measurement may lead to the wrong conclusion.

Instead of just looking at one trace we would like to have many traces which we can compare to remove noise and see a difference between high and low. That we have to look at (near) location 2500 and that the value is high is not very clear from the blue trace alone but when we take the difference with the green trace, where the value is low, this becomes obvious. By using multiple 'blue' (high) and 'green' (low) traces we can also average out at least some of the measurement noise. Of course a bit of the key will always be 'low' or 'high' for all traces that we generate from this device; we only get blue or green traces (and we won't know which or else we did not need to do this attack). We have to use a more indirect route to extract useful information using this approach.



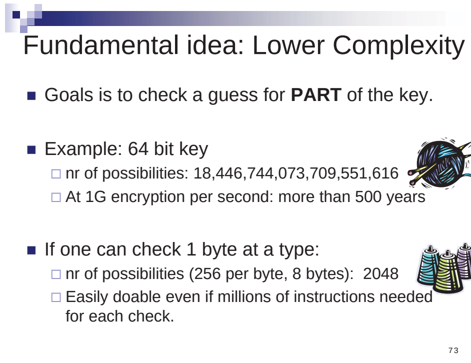
DPA: Bit Propagation

- Collect Traces (random inputs I1,I2,...In)
- Choose input bit *X*
- Group traces by value bit *X*
- Difference trace shows where bit *X* used

Input
I1 I2 ... In
Trace
T1 T2 ... Tn
I1 bit X = 1
T1 => G1
I2 bit X = 0
T2 => G0
etc...
T1
T2
G1
G0
Avg – Avg
Diff. trace
Any function (#1s)

Partial diff. traces for DES input bit *X* = 16,17,18,19

72



Fundamental idea: Lower Complexity

- Goals is to check a guess for **PART** of the key.
- Example: 64 bit key
  - nr of possibilities: 18,446,744,073,709,551,616
  - At 1G encryption per second: more than 500 years
- If one can check 1 byte at a type:
  - nr of possibilities (256 per byte, 8 bytes): 2048
  - Easily doable even if millions of instructions needed for each check.

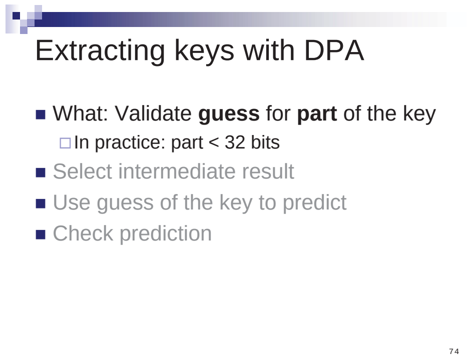
73

As a first step we will use are differential method to find the locations in the computation at which certain bits of the input are being used. We need to generate many high (blue-bit is 1) and low (green-bit is 0) traces. We can do this be choosing random inputs $I_1, \ldots, I_n$, feeding these to the card and recording the power traces that this gives us. If bit X of the input I_j is 1 we put the trace in group G1, otherwise we put it in group G0. We end up with two groups containing approximately $n/2$ each traces. Where ever input bit X is used in the computation, G1 will be working with a one while G0 with a zero; G1 will on average have a higher power consumption at this location. Thus by subtracting the average of the traces in G0 from the average of the traces in G1 we should get a positive value at this time location. For all values that do not depend on bit X the expected values in groups G1 and G0 are the same; the difference in averages should thus be (close to) zero.

Spikes in the difference trace (the difference between the averages of the trances) thus show where the bit we selected is being used. We have thus found a way to extract information by differential information. The next step is to extract (even) more interesting information. The first observation that we can make is that we do are not restricted to checking for an input bit; we can look for any boolean function that we can compute (given the input). A second observation is that the function does not even have to be boolean; we can look for an arbitrary function and group based on 'the function outcome has many ones' (so expect high power consumption where this is computed)' or 'the function outcome has few ones (so expect low power consumption where this is computed)'. Again the difference between the two group will reveal where this function outcome is being computed through a spike in the difference trace.

So how can we use the knowledge whether a function is being computed to extract keys? The idea is that with this check we can validate a guess for part of the key, for example one byte. If we can do this then we reduce the number of guesses needed from exponential to linear; e.g. for a 64 bit key we would need to only check a few thousand options rather than billions of billions of possibilities. This is thus worthwhile even if checking making a single check is a significant amount of work.

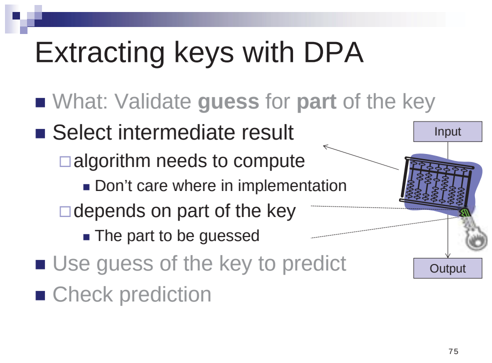
Our goal is thus to check a guess for part, for example one byte, of the key. We do this by finding an intermediate result that will have to be computed. For example, an implementation of AES will have to, at some point, compute the xor of the (round)key and the input. If we take one byte of this, e.g. (input[1] XOR key[1]) then this depends only on one byte of the (round)key and information that we know (the input).



Extracting keys with DPA

- What: Validate **guess** for **part** of the key
  - In practice: part < 32 bits
- Select intermediate result
- Use guess of the key to predict
- Check prediction

74



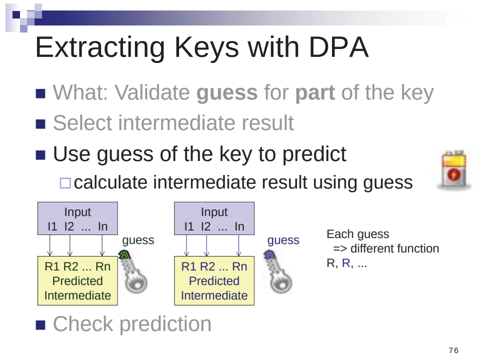
Extracting keys with DPA

- What: Validate **guess** for **part** of the key
- Select intermediate result
  - algorithm needs to compute
    - Don't care where in implementation
  - depends on part of the key
    - The part to be guessed
- Use guess of the key to predict
- Check prediction

Input
Output

75

Our targeted intermediate result depends only on the input and a small part of the key. If we knew this part of the key we could thus use the differential approach above to find out where the intermediate result is computed: where it is the difference trace will show a spike. However, if we use a wrong value for the key then the result that we predict will not actually be computed so no spike will occur anywhere. But wait - this is exactly what we are looking for; some way to tell whether our guess is correct. If the guess is correct there will be a spike somewhere in the difference trace (we don't even care where) and if it is not then there will be no such spike. (There may actually be some smaller spikes even for incorrect guesses as the 'wrong' result may still be correlated to some computed value but the spike for the correct guess will be the biggest.)

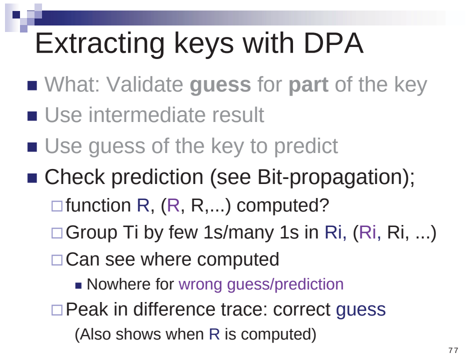


Extracting Keys with DPA

- What: Validate **guess** for **part** of the key
- Select intermediate result
- Use guess of the key to predict
  - calculate intermediate result using guess
- Check prediction

Input
I1 I2 ... In
guess
R1 R2 ... Rn
Predicted
Intermediate
Input
I1 I2 ... In
guess
R1 R2 ... Rn
Predicted
Intermediate
Each guess
=> different function
R, R, ...

76

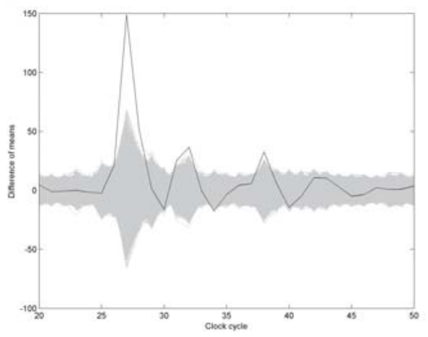


Extracting keys with DPA

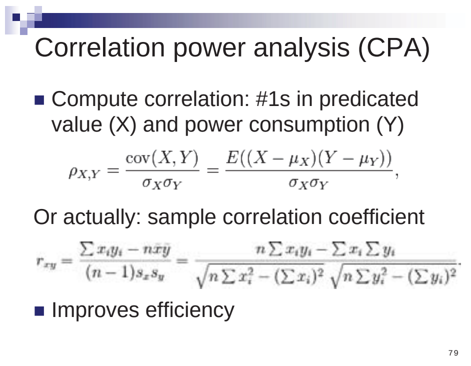
- What: Validate **guess** for **part** of the key
- Use intermediate result
- Use guess of the key to predict
- Check prediction (see Bit-propagation);
  - function R, (R, R,...) computed?
  - Group Ti by few 1s/many 1s in Ri, (Ri, Ri, ...)
  - Can see where computed
    - Nowhere for wrong guess/prediction
  - Peak in difference trace: correct guess
    (Also shows when R is computed)

77

A guess for this small part of the key can thus be checked as follows: We try each possible guess for the part of the key. Each guess gives us a different function from input to intermediate result. Only the function for the correct guess is actually computed by the device so it will be the only one that will show up by producing a spike in the difference trace.



Difference trace for Correct guess and incorrect guesses



Correlation power analysis (CPA)

- Compute correlation: #1s in predicated value (X) and power consumption (Y)

$$\rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X\sigma_Y} = \frac{E((X-\mu_X)(Y-\mu_Y))}{\sigma_X\sigma_Y},$$

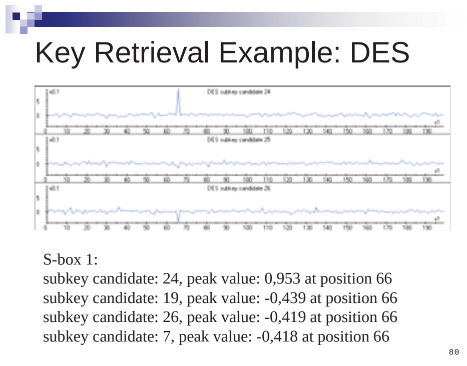
Or actually: sample correlation coefficient

$$r_{xy} = \frac{\sum x_i y_i - n\bar{x}\bar{y}}{(n-1)s_x s_y} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\,\sqrt{n\sum y_i^2 - (\sum y_i)^2}}.$$

- Improves efficiency

79

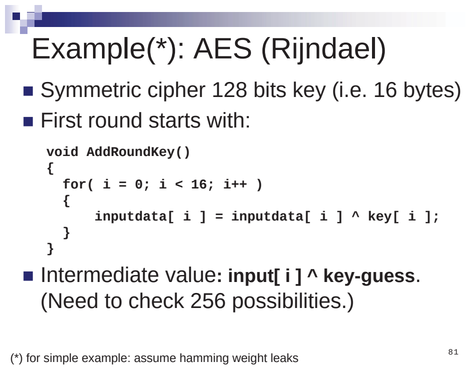
We so far look at a single 'low' group and a single 'high' group. This discards some potentially useful information (e.g. should the power consumption be high (6 ones) or very high (8 ones)). We can actually be slightly more efficient by using this information in our analysis: instead of only considering low and high we can consider all possible values; e.g. no ones up to 8 ones when looking at a byte. Group 0 should have the lowest power consumption, 1 a little higher, 2 higher again etc. By computing the correlation between the group number and the power consumption we get a score which shows an even bigger difference between a correct and an incorrect guess. (This is important if the differences in power usage are small in comparison to the amount of noise in the measurements.)



Key Retrieval Example: DES

S-box 1:
subkey candidate: 24, peak value: 0,953 at position 66
subkey candidate: 19, peak value: -0,439 at position 66
subkey candidate: 26, peak value: -0,419 at position 66
subkey candidate: 7, peak value: -0,418 at position 66

80



Example(*): AES (Rijndael)

- Symmetric cipher 128 bits key (i.e. 16 bytes)
- First round starts with:

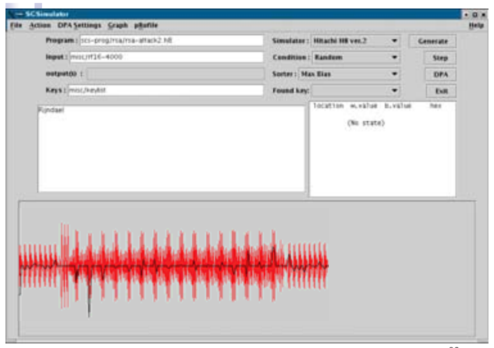
```
void AddRoundKey()
{
  for( i = 0; i < 16; i++ )
  {
      inputdata[ i ] = inputdata[ i ] ^ key[ i ];
  }
}
```

- Intermediate value: **input[i] ^ key-guess**. (Need to check 256 possibilities.)

(*) for simple example: assume hamming weight leaks

81

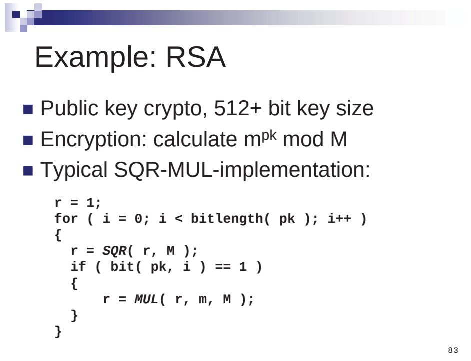
The results show some figures of difference traces for different key guesses with the correct key guess clearly standing out. The image of the simulation tool below shows the difference trace for the correct key guess projected on top of the simulated power consumption (for the AddRound-Key method only). Here we are attacking the second byte of the round key here and the spike occurs in the second iteration of the loop as expected.



Program : scs-prog/rsa/rsa-attack2.h8
Input : misc/rf16-4000
output(s) :
Keys : misc/keylist
Simulator : Hitachi H8 ver.2
Condition : Random
Sorter : Max Bias
Found key :
Rijndael
location w.value b.value hex
(No state)

Simulation tool from the PINPAS Project

82



Example: RSA

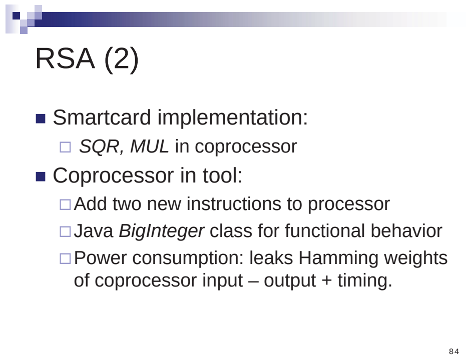
- Public key crypto, 512+ bit key size
- Encryption: calculate m^{pk} mod M
- Typical SQR-MUL-implementation:

```
r = 1;
for ( i = 0; i < bitlength( pk ); i++ )
{
  r = SQR( r, M );
  if ( bit( pk, i ) == 1 )
  {
      r = MUL( r, m, M );
  }
}
```

83

Consider a SQR-MUL implementation of RSA. In such an implementation the exponentiation (message M to the power key) that needs to be done in RSA encryption/decryption is performed by doing a sequence of square and multiply operations. Squaring a number is multiplying its exponent by 2 (i.e. shifting it one position to the left). Multiplying by M is adding 1 to the exponent (i.e. setting the least significant bit). This gives an easy iterative way to build M^k.

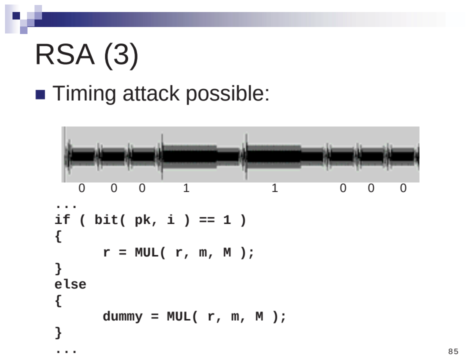
Unfortunately this implementation is very vulnerable to a power analysis-timing attack: if we can recognize the different rounds of SQR and MUL operations from the power consumption (which is likely as MUL is harder than SQR) such an implementation basically allows reading out the key from the power consumption graph as the simulation clearly shows.



RSA (2)

- Smartcard implementation:
  - *SQR*, *MUL* in coprocessor
- Coprocessor in tool:
  - Add two new instructions to processor
  - Java *BigInteger* class for functional behavior
  - Power consumption: leaks Hamming weights of coprocessor input – output + timing.

84



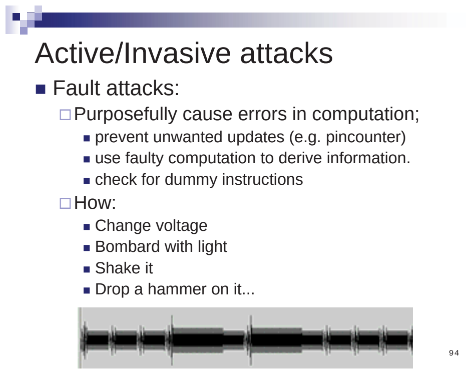
RSA (3)

- Timing attack possible:

```
...
if ( bit( pk, i ) == 1 )
{
      r = MUL( r, m, M );
}
else
{
      dummy = MUL( r, m, M );
}
...
```

85

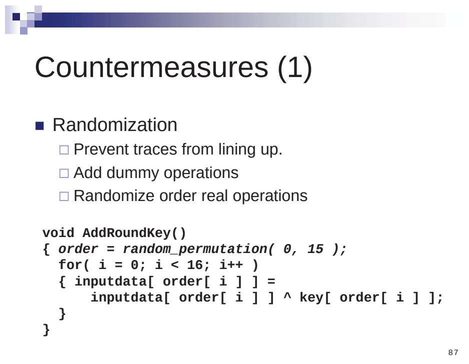
One defense against timing attacks is to make the running time independent of the data being used. Here this can be achieved by inserting a dummy multiplication; at the cost of extra operations (performance loss) the revealing pattern is removed.



Active/Invasive attacks

- Fault attacks:
  - Purposefully cause errors in computation;
    - prevent unwanted updates (e.g. pincounter)
    - use faulty computation to derive information.
    - check for dummy instructions
  - How:
    - Change voltage
    - Bombard with light
    - Shake it
    - Drop a hammer on it...

94



Countermeasures (1)

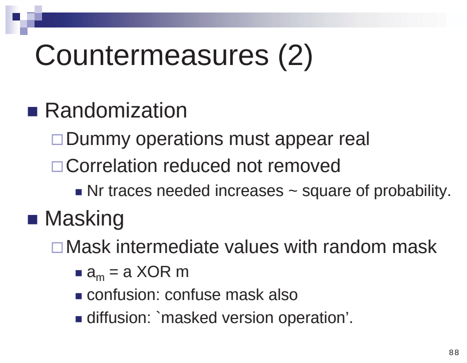
- Randomization
  - Prevent traces from lining up.
  - Add dummy operations
  - Randomize order real operations

```
void AddRoundKey()
{ order = random_permutation( 0, 15 );
  for( i = 0; i < 16; i++ )
  { inputdata[ order[ i ] ] =
      inputdata[ order[ i ] ] ^ key[ order[ i ] ];
  }
}
```

87

We have to make sure that the ‘dummy operation’ looks and act real; if they are obvious the attacker can remove them. One way to test for dummy operations is to induce errors. If the end result remains unchanged the attacker knows that the error occurred in a dummy calculation. In the RSA implementation with timing countermeasure this can be used to find which are MUL operations are dummies, i.e. which bits of the key are zero.

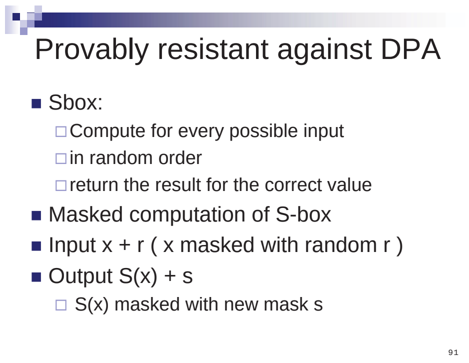
Removing data dependent timing information does not stop DPA attacks. Actually, there preventing traces from lining up is a possible defense; if the same operation in different runs happens in different places we cannot compare the power consumption between different traces anymore.



Countermeasures (2)

- Randomization
  - Dummy operations must appear real
  - Correlation reduced not removed
    - Nr traces needed increases ~ square of probability.
- Masking
  - Mask intermediate values with random mask
    - a_m = a XOR m
    - confusion: confuse mask also
    - diffusion: `masked version operation'.

88



Provably resistant against DPA

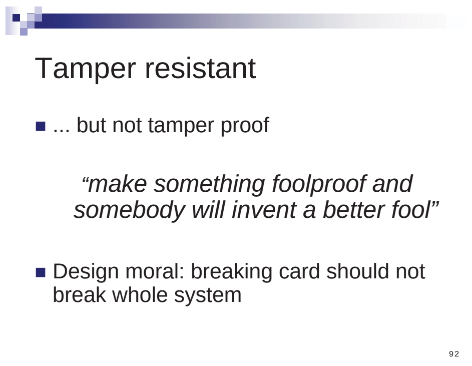
- Sbox:
  - Compute for every possible input
  - in random order
  - return the result for the correct value
- Masked computation of S-box
- Input x + r (x masked with random r)
- Output S(x) + s
  - S(x) masked with new mask s

91

One can also try to prevent the comparison between traces by changing the data rather than the operations. By masking the data computed with a different random mask in each run comparing different traces no longer works. One way of masking is by using an XOR with a random value.

Note that we cannot remove the mask to preform operations such as S-Boxes (this would create an attackable intermediate result); instead we create a version of the S-box that works for this mask. Only once the final result is ready will the mask be removed. (The final result will go to the output anyway so no need to mask it.)

Schemes to implement key components S-boxes with combinations of masking and randomization that make them provably resistant to DPA have been proposed. However, the performance overhead is big and, though resistant to DPA, they are not necessarily resistant to other types of power analysis attacks.



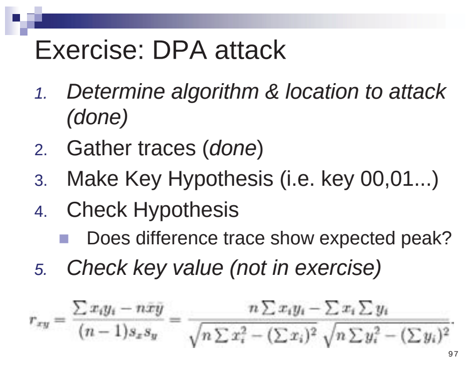
Tamper resistant

- ... but not tamper proof

"make something foolproof and somebody will invent a better fool"

- Design moral: breaking card should not break whole system

92



Exercise: DPA attack

1. *Determine algorithm & location to attack (done)*
2. Gather traces (*done*)
3. Make Key Hypothesis (i.e. key 00,01...)
4. Check Hypothesis
   - Does difference trace show expected peak?
5. *Check key value (not in exercise)*

$$r_{xy} = \frac{\sum x_i y_i - n\bar{x}\bar{y}}{(n-1)s_x s_y} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\,\sqrt{n\sum y_i^2 - (\sum y_i)^2}}.$$

97

A main message to take away is that tamper resistant is not tamper proof. Often this is good enough; it is not the weakest point in the system but the tamper resistant device should not be a single point of failure. E.g. using the same key on all devices would likely be a very bad idea.

6.5 Exercises

1. Consider again the online music store of the previous two chapters. Review your requirements analysis, considering the role of identity management and authentication; gather threats and countermeasures considering their effect on (these and other) threates as well as the security goals.
2. See separate document: DPA exercise.
3. A differential power analysis attack targets an intermediate value that the algorithm needs to calculate and which depends on part of the key. It tests a hypothesis about a part of the key by comparing the predicted outcome using this key with the actual power consumption of the device.
   (a) Why is this useful; how can it be used to reduce the complexity of an attack?
   (b) Assuming the algorithm and the input given to the algorithm are known, which instruction(s) in the following piece of pseudo code would be useable for a DPA attack (explain your answer).
   ```
   1. if i > 15 go to 8;
   2. r1 :=  key[i];
   3. r1 := input[i] XOR r1;
   4. r1 := SBOX[r1];
   5. output = output XOR r1;
   6. i := i+1;
   7. go to 1;
   8. return output;
   ```
   (c) The code in previous part contains a loop. In which iteration(s) would it be easiest to attack the algorithm?

4. With pin payments two factor authentication of the paying party is performed.
   (a) Which are the two factors, how are they implemented in this example and why are two factors used rather than only one?
   (b) What is the third factor and how could it be applied in this setting?
   (c) After entering the pin the terminal shows the amount to be transfered and asks for confimation from the user. What type of attacks is this ment to prevent and how effective is this method?
5. The privium program at Schiphol airport allows its users to avoid the passport check by using iris scans: " Ĕ The same advantage applies at the border check, which is directly followed by security. Ĕ The identity is determined by the unique characteristics of the iris, which are stored, neatly encrypted, on the membership card. Ĕ The whole process takes about 30 seconds and then Ű if everything checks out Ű a gate opens and the security check of personal items and hand luggage follows. If things do not check out (for example because the iris is not correctly recognized) the traveller can continue to the Śold fashionedŠ passport check. Ĕ in the US, experiments with palm prints have been taking place for several years already, however that system has quite a few technical difficulties. The biometrics approach of Schiphol can be considered innovative: the accuracy of the iris-scan is much higher than other forms, such as the before mentioned palm print recognition and no other system lets the clients carry their own data with them. This last point is fundamental. ŞWe have had to develop every thing ourselvesŤ says Conny Lanza, ŞAll existing technology assumes that data is stored in a database and we did not want thatŤ. [wereldverkenning.nl, 2000]
   (a) Why have the developers chosen to store biometric profiles on a smartcard instead of a central database; give advantages and risks for both approaches. How could you try to minimize the risks you mentioned?
   (b) Sketch a graph with the false accept and false reject rates for iris and palm print recognition.
   (c) Biometric measurements are never precisely the same and a threshold need to be chosen for allowed deviation. Based on false accept and false reject rate explain what threshold you would use in this application and indicate this threshold in the sketch of part b.