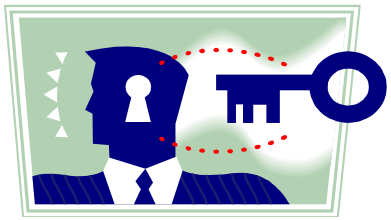


Cryptographic Message Authentication

Entity Authentication Passwords, Challenge-Response

+Time Stamping

Nicolas T. Courtois
- University College London

Wen 20 NOV or Thur 21? VOTE!

LITTLE EXAM = QUIZ

15%, Moodle-based,
about 45 mins

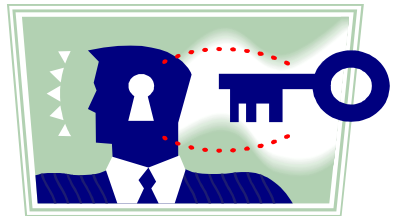
- Q/A like “Bell La Padula”
- Multiple Choice
  - 1 or several answers are valid

PLEASE BRING YOUR LAPTOP or tablet.

Two Main Areas in Authentication

1. Cryptographic Message Authentication
   - MACs / Digital signatures + complex protocols
2. Entity Authentication,
   - Passwords,
     - static = bad
   - Challenge-Response:
     - dynamic:
       the right answer to all questions at the exam ☺

Vocabulary



Basic concepts.

1. Identification: declare who you are.
2. [Entity] Authentication: prove it.

But a Secure Identification Scheme = 1+2 =
Entity Authentication Scheme
can be considered as synonyms.

Crypto Revision

Goals of Cryptography

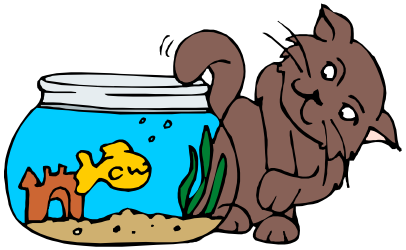
1. ~~Confidentiality~~:

2. Authenticity,
   Integrity, Non-repudiation…

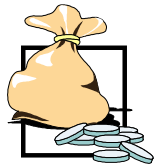
**The Security ? 3-point Formal Approach

What is Security ? Inability to achieve:

1. Security against what: Adversarial Goal.



2. Against whom: resources of the Adversary: money, human resources, computing power, memory, risk, expertise, etc..

3. Access to the system.

***The Security ? 3-point Formal Approach

Security Notion / Definition = a triple:

1. Adversarial Goal.
2. Resources of the Adversary.
3. Access / Attack.

One can ONLY talk about security w.r.t. a given triple. May not hold for other triple.

Authenticity -Vocabulary

Two Main Areas:

1. Message Authentication.
2. Entity Authentication / Identification

Closely related…

Entity Authentication / Identification

3 FACTORS:

A person/device can be authenticated by

1. Something that he/it knows.
   - PIN, password, knowledge of an AES key, private RSA key etc..
2. Something that he/it has.
   - Smart card, USB key, TPM module, and other tamper-resistant hardware…
3. Something that he/it is.
   - Biometrics, unique physical characteristics (cf. snow flake).

Multi-factor authentication:

To enter the office, one needs:

1. A PIN.
2. A smart card.

We speak about 2-factor system.

High security systems (e.g. bank vault, military lab, etc.) requires to systematically and simultaneously use 3 factors

=> Good security.

Bad Security Advice [5 March 2015]

dailymail.co.uk/sciencetech/article-2980880/Death-SIM-card-Expert-believes-passwords-replace-ic

Death of the SIM card? Expert believes passwords should replace iconic microchips

- Dr Markus Kuhn said the SIM could have been replaced a long time ago
- He believes one alternative could be typing in a user identifier and password directly into a phone - like we do with Wi-Fi networks

Dr Markus Kuhn from the University of Cambridge said the SIM (stock image) could have been replaced long ago. He believes alternatives include typing in a user identifier and password

Message Authenticity – Goals

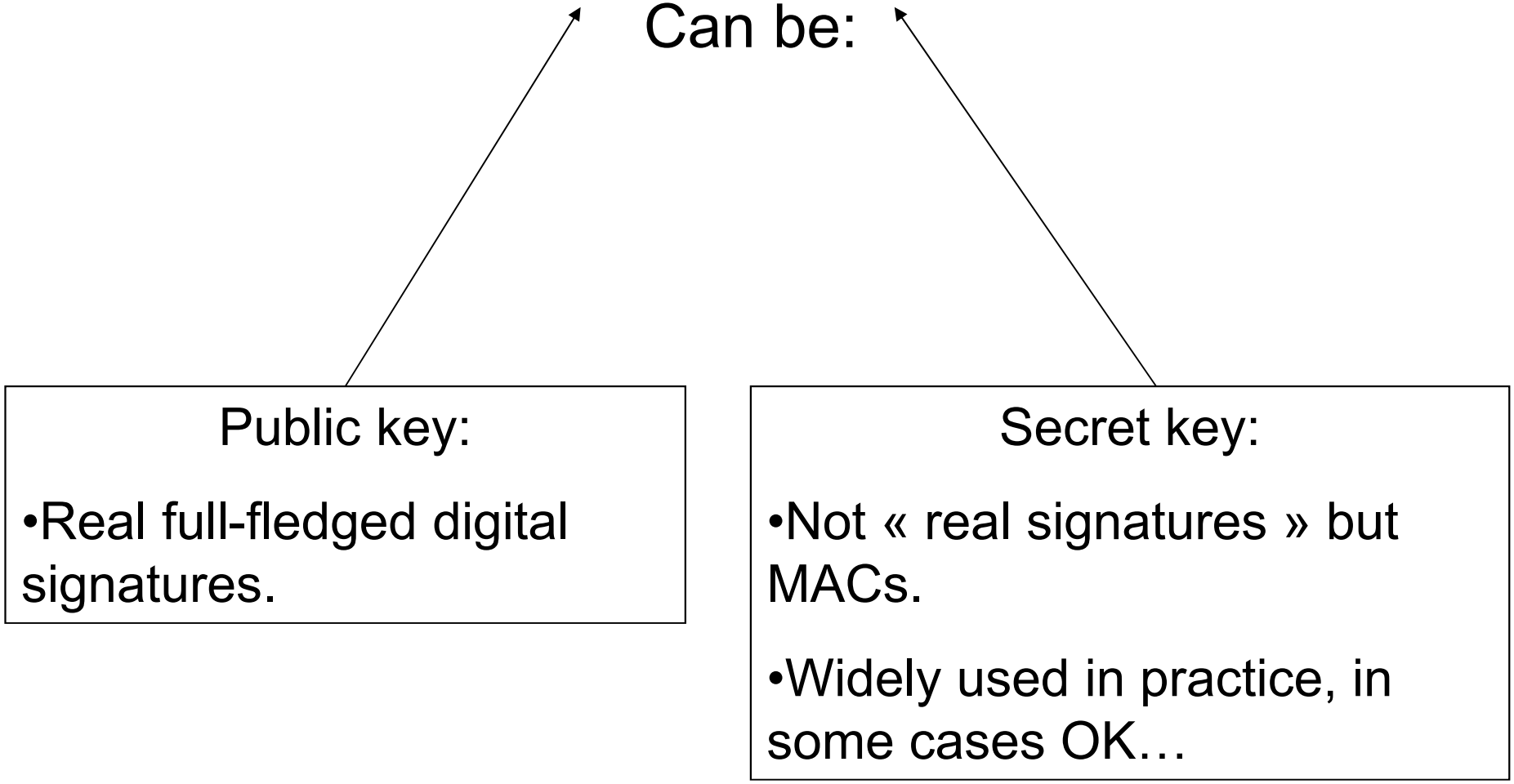
Different security levels:

1. Correct transmission – no (random) transmission error. A malicious attacker can always modify it.
   - Achieved with CRC and/or error correction/detection codes.
2. Integrity – no modification possible if the "tag/digest" is authentic. If we cannot guarantee the authenticity of the tag, a malicious attacker can still modify and re-compute the hash.
   - Achieved with cryptographic hash functions (= MDC). (e.g. SHA-1).
3. Authenticity – specific source. Authentified with some secret information (key).
   - Achieved with a MAC (= a hash function with a key = a secret-key signature).

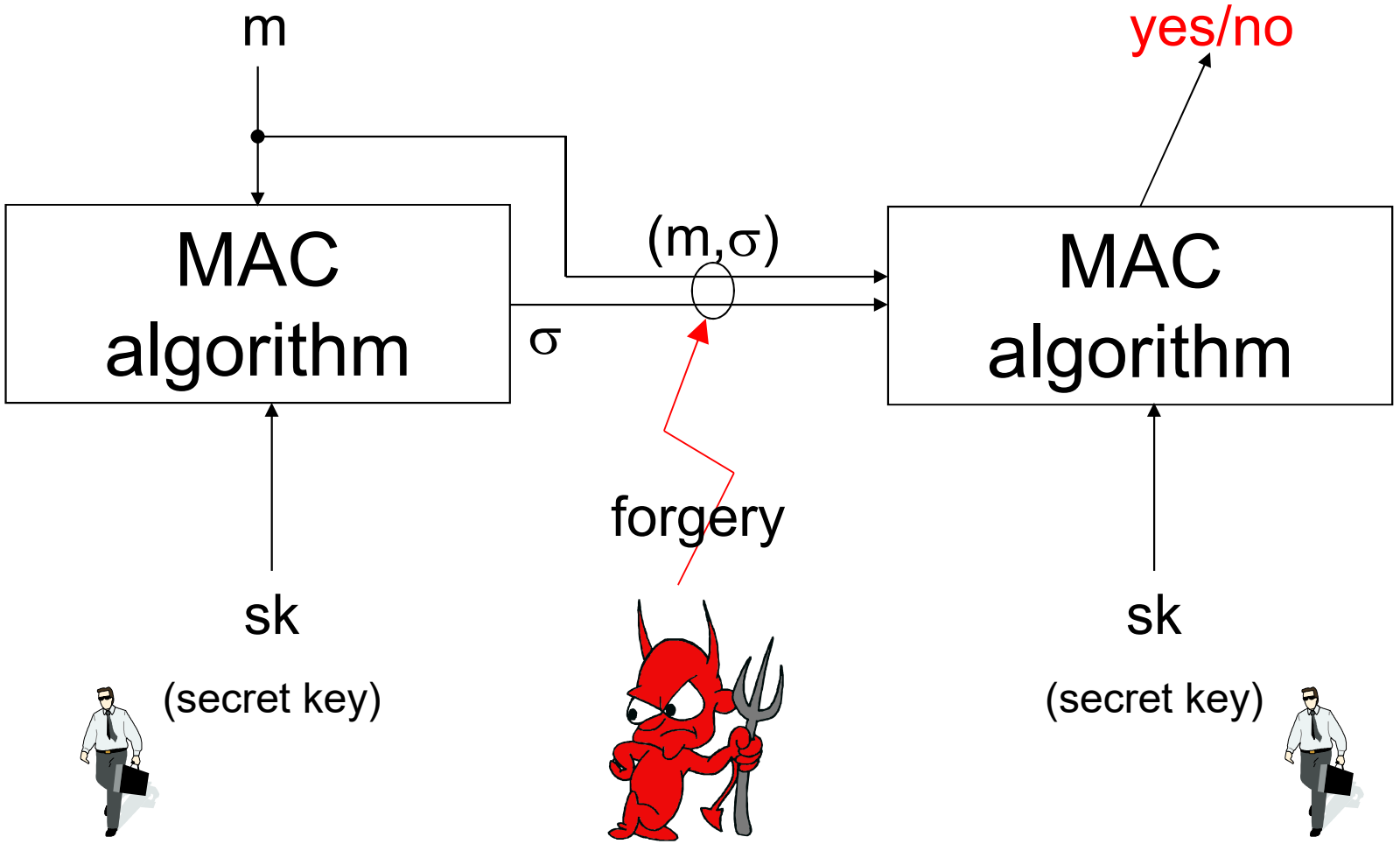
4a. Non-repudiation – very strong requirement. Only one person/entity/device can produce this document.
   - Achieved with Digital Signatures. The strongest method of message authentication.

4b. Public verify-ability. Everybody can be convinced of the authenticity (trust the bank ?).
   - Achieved with Digital Signatures. The strongest method of message authentication.

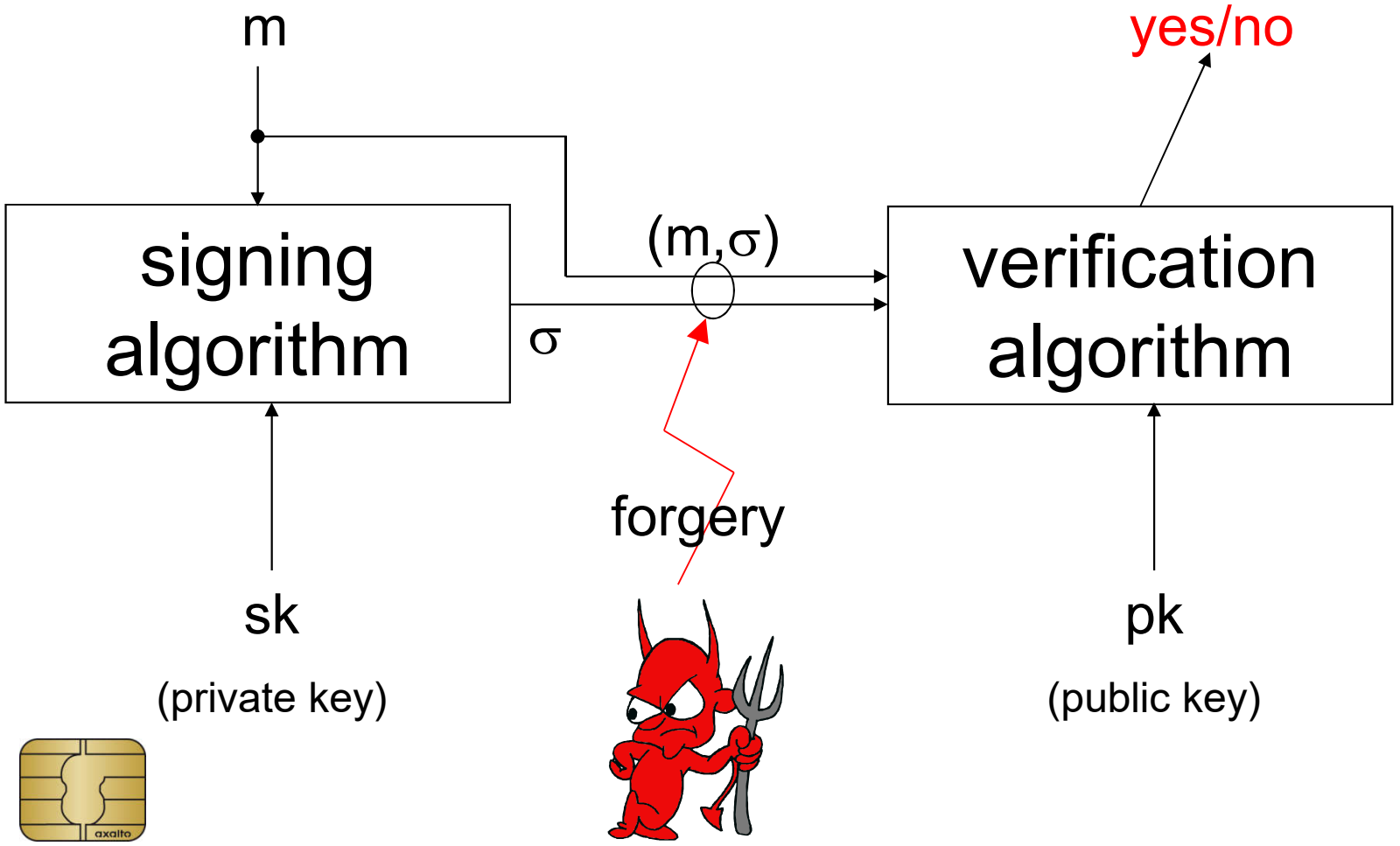
Signatures


Can be:
Public key:
•Real full-fledged digital signatures.
Secret key:
•Not « real signatures » but MACs.
•Widely used in practice, in some cases OK…

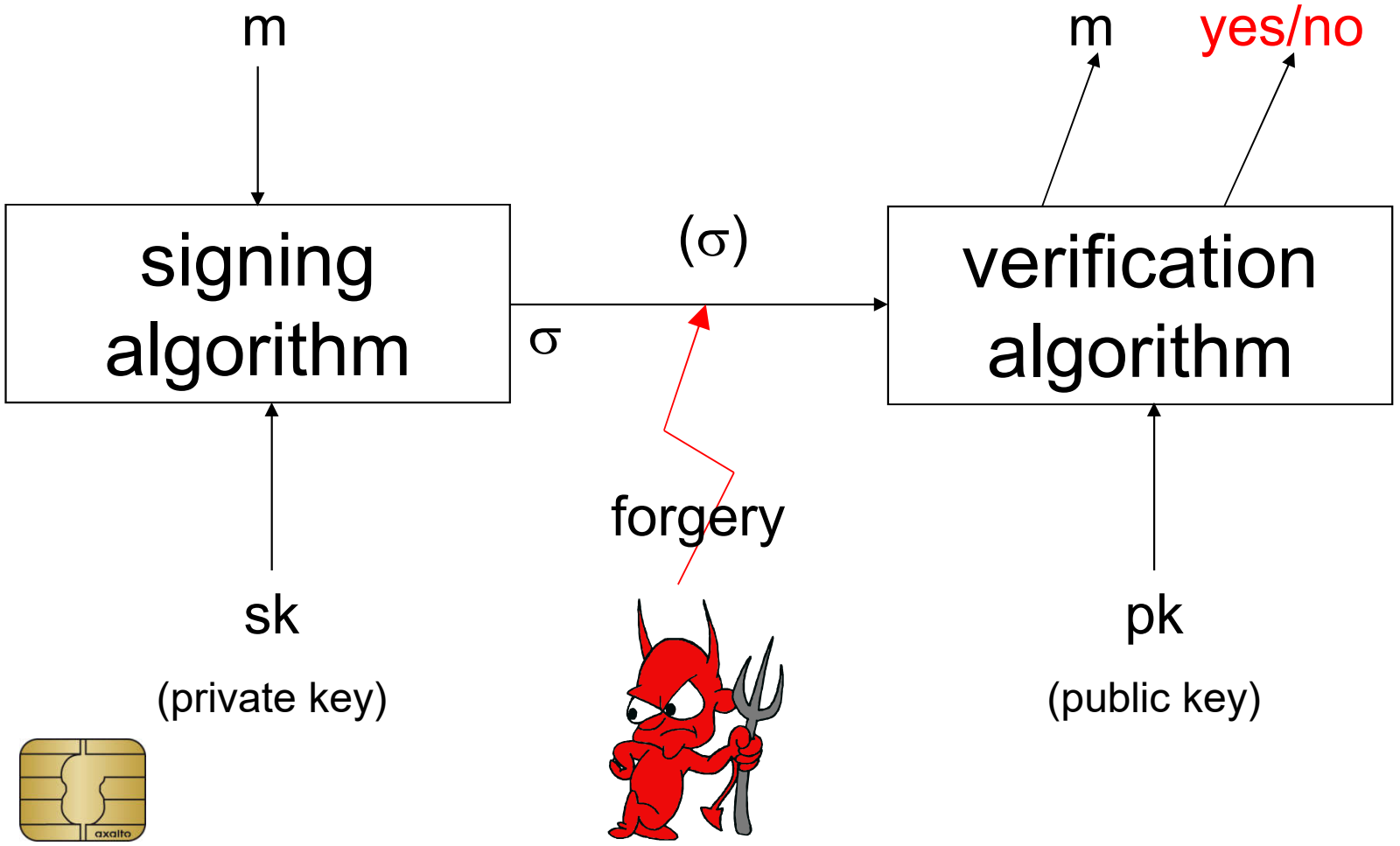
MACs = "Secret-Key Signatures"



Digital Signatures



*Digital Signatures with Message Recovery



Signatures - Requirements

1. Authenticity – guarantees the document signed by…
2. Non-repudiation – normally only possible with public-key signatures.
   - Unless if we assume that we dispose of a tamper-resistant hardware (e.g. a smart card) the non-repudiation can be achieved with a MAC based on AES !
3. Public verify-ability - normally only possible with public-key signatures.
   - Unless there is a trusted third party (e.g. independent and trusted authority, an electronic notary service), then public verify-ability will be achieved with a MAC based on AES !

CONCLUSION; secret key signatures can work in practice… but are fundamentally either less secure or less practical (what if the notary stops responding, the smart card destroys itself because it thinks it is being attacked etc..).

Digital Signatures: Top of the Top:

- The strongest known form of Message Authentication:

*Digital Signatures vs. Authentication

- Strongest known form of Message Authentication.
- Allows also authentication of a token/device/person (e.g. EMV DDA, US Passport):
  - challenge –response (just sign the challenge)
- The reverse does not hold:
  - Not always possible to transform authentication into signature. More costly in general !

Sym. encryption << P.K. authentication < signature



Part 3

Cryptographic Hashing

Hash Functions

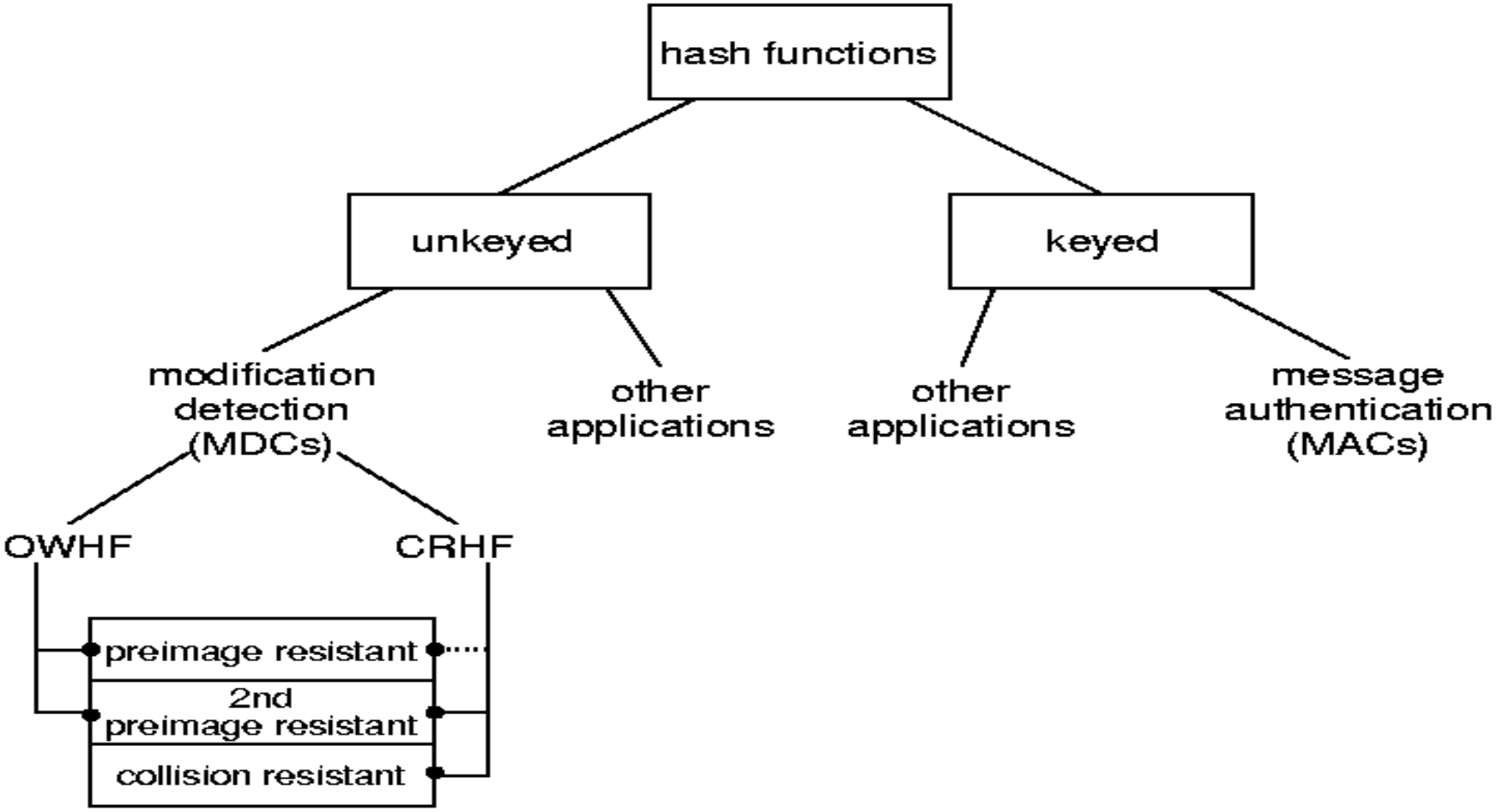


Figure 9.1: *Simplified classification of cryptographic hash functions and applications.*

What do We Sign ? The Problem:

Public key crypto is very slow.

Sign a long message with RSA, impossible, even on a 40 GHz CPU !

⇒Use hash function.

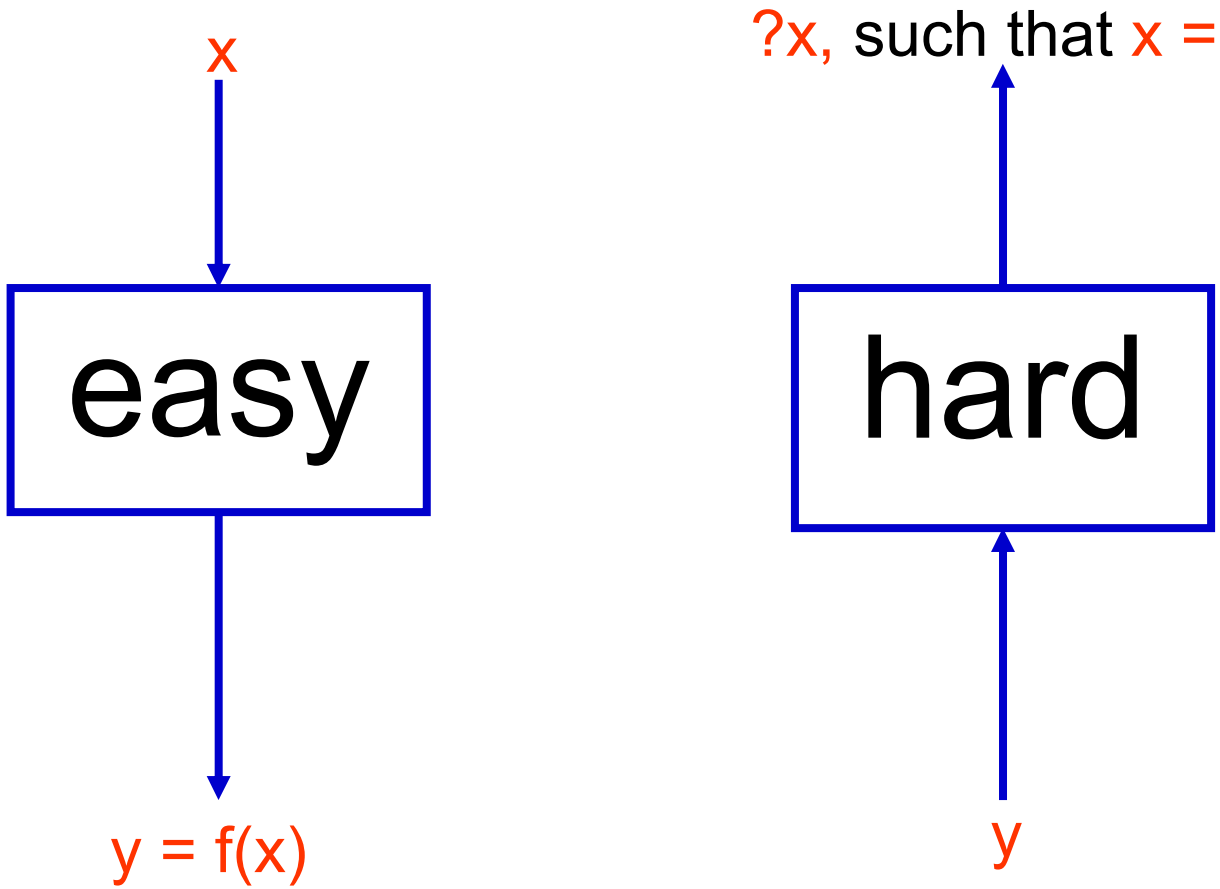
⇒Sign a short « digest » of the message.

Hashing

In computer science we have:

- hashing (weak), not security just some mixing and chopping…
  - must be very fast.
  - Example: hash tables, such as hash_set<> in C++ STL.
- cryptographic hashing (strong),
  - nobody should ever find any weakness in it
  - should be very fast, but NOT at the expense of security !

One-Way Functions (OWF)

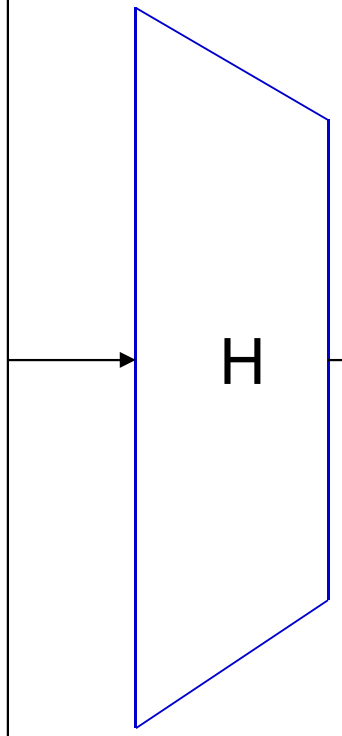


[Cryptographic] Hash Function:

m

A hash function (or hash algorithm) is a reproducible method of turning data (usually a message or a file) into a number suitable to be handled by a computer. These functions provide a way of creating a small digital "fingerprint" from any kind of data. The function chops and mixes (i.e., substitutes or transposes) the data to create the fingerprint, often called a hash value. The hash value is commonly represented as a short string of random-looking letters and numbers (Binary data written in hexadecimal notation).

0-∞ bits



H(m)

A94A8FE5
CCB19BA6
1C4C0873
D391E987
982FBBD3

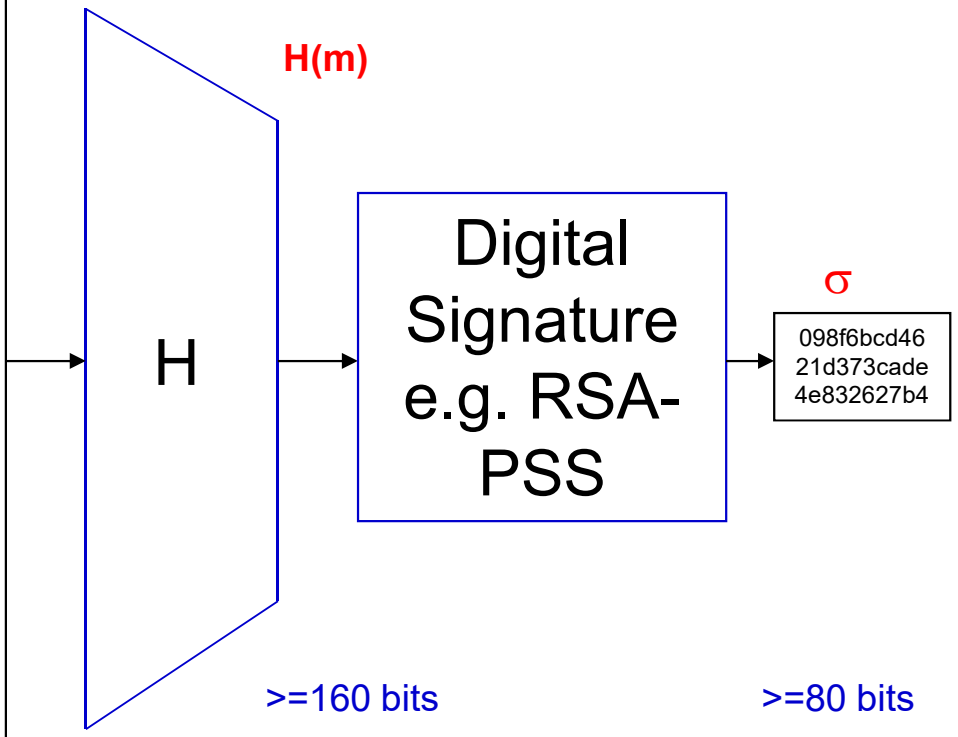
>=160 bits

Hash-then-Sign

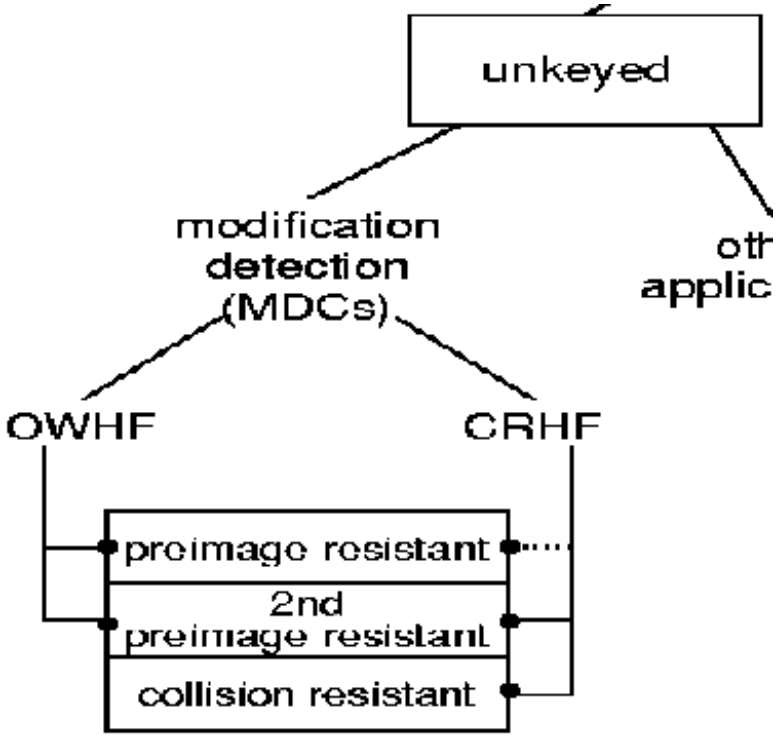
m

A hash function (or hash algorithm) is a reproducible method of turning data (usually a message or a file) into a number suitable to be handled by a computer. These functions provide a way of creating a small digital "fingerprint" from any kind of data. The function chops and mixes (i.e., substitutes or transposes) the data to create the fingerprint, often called a hash value. The hash value is commonly represented as a short string of random-looking letters and numbers (Binary data written in hexadecimal notation).

0-∞ bits



Hash Functions = MDC

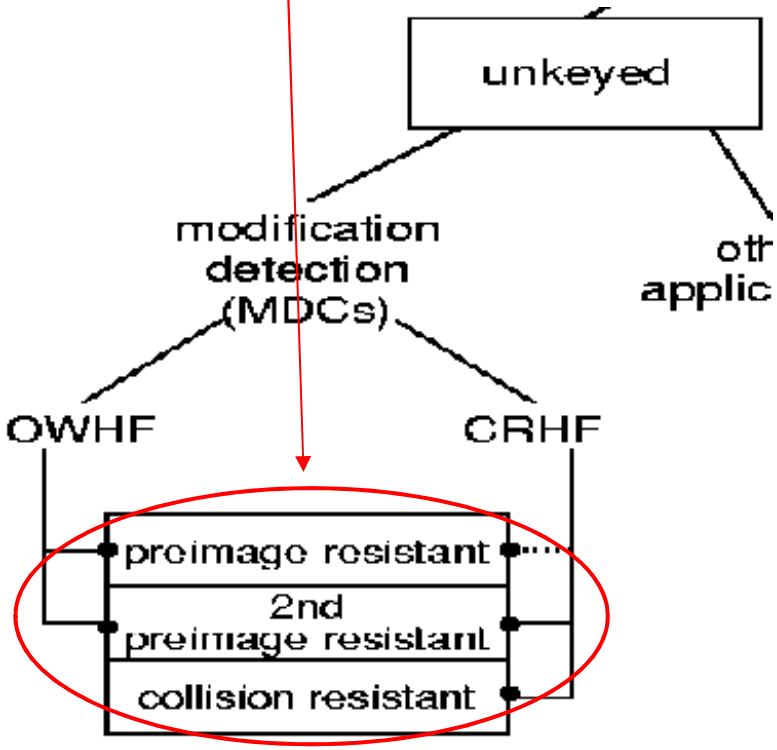


OW= One Wayness

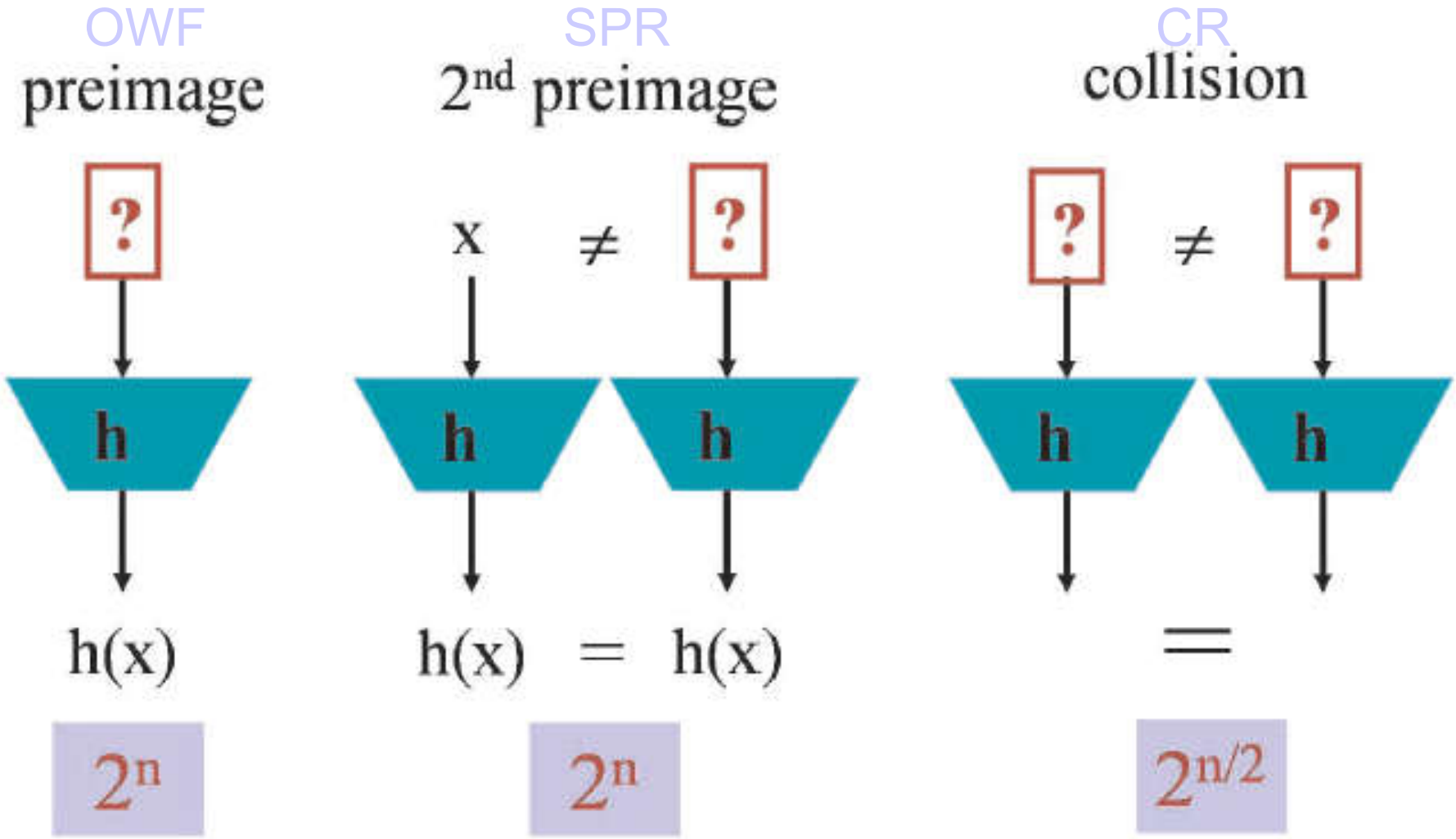
CR= Collision Resistance

Requirements

- public function, no secret keys or parameters.
- arbitrary (or very long) length -> fixed length
- easy/fast to compute
- hard to:



Requirements



Cryptographic Hash Functions

Hash functions – typical requirements:

- OWHF = One-Way Hash Functions. Strict Minimum
  - OWF
  - SPR = Second Preimage Resistant
- CRHF = Collision-Resistant Hash Functions. A Lot / 2 little ?
  - OWF
  - CR
- Many people demand even much more of hash functions:
  - OWF
  - SPR
  - CR
  - PRF – very strong requirement. Pseudo Random Function
  - very fast, standardized, with partial security proofs etc.

Preimage Resistance == OWF

OWF = Preimage Resistant:

Let y be chosen at random.

"Hard" to find x s.t. H(x)=y.

Hard=? - Concrete security:

Let y be on n bits.

- **It should take time about 2^n.**
- Remark: If it takes $2^{n/3}$ it is a OWF in asymptotic sense, yet very insecure in practice !

Note: OW seems quite easy to achieve.

Another Important Requirement

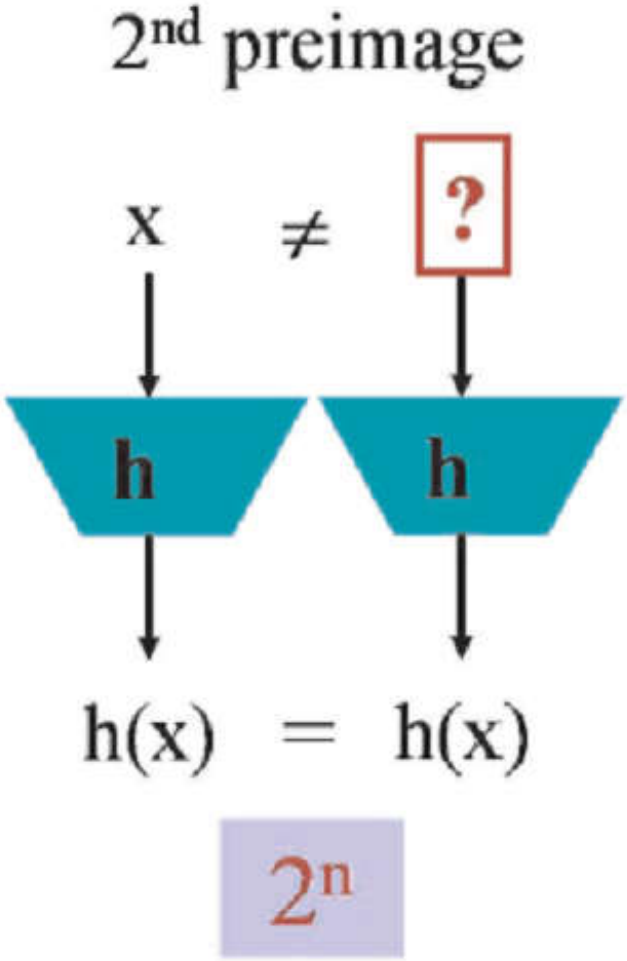
SPR – Second Preimage Resistant.

Note: Seems very feasible to achieve.

Hard=? - Concrete security:

- **It should take time about 2^n.**

- Knowing one x can helps to reduce the difficulty if there is a weakness somewhere…
- For a well designed function, to know one x doesn't seem to help a lot…



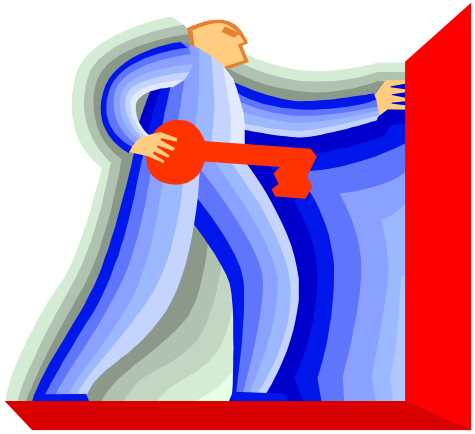
Passwords

The Key Idea

Prover sends a password to a Verifier.

The channel is assumed private.

- Integrity?
  - The channel doesn't really have to be authenticated or noise-free…
    - this will affect usability and availability, but not the security

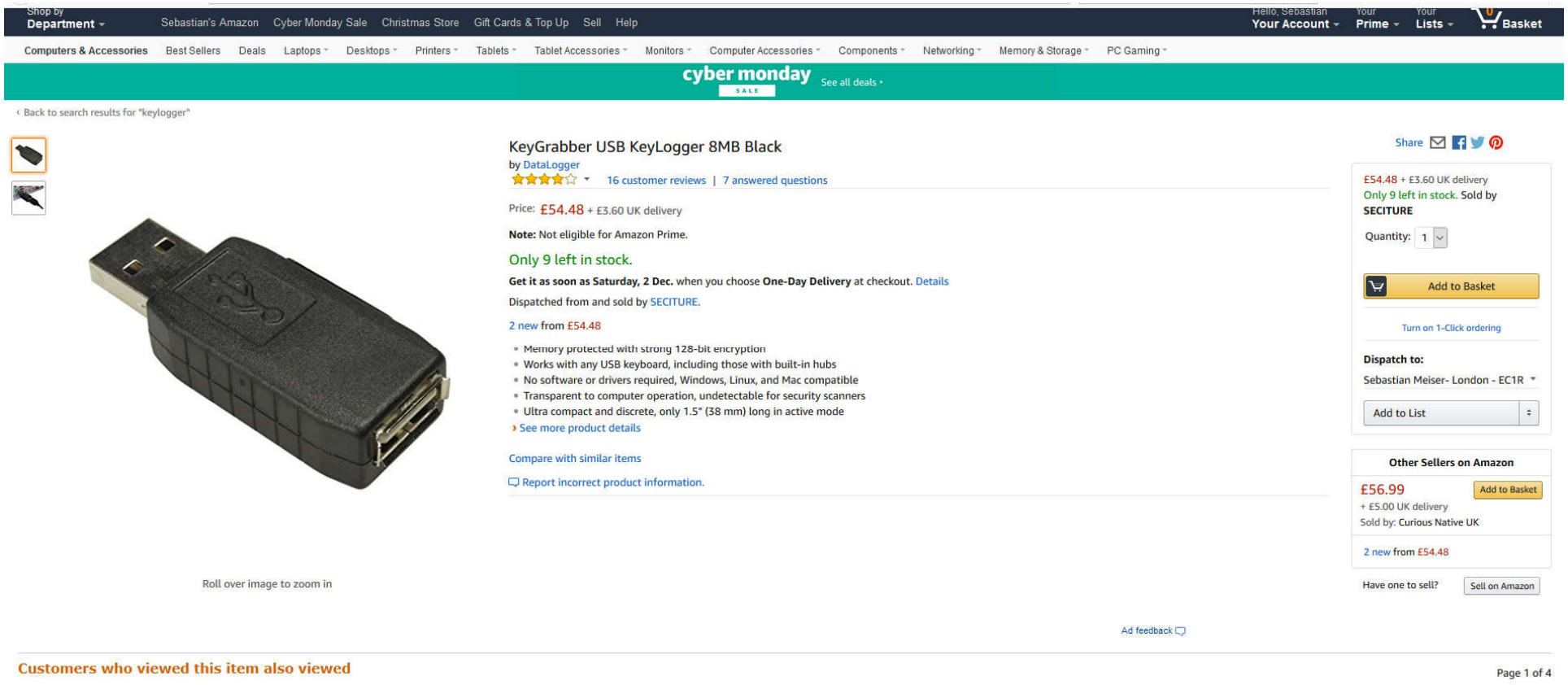


Areas of Study

Care is needed when:

- Choosing the password
  - (and the technology: e.g. visual passwords)
- Storing the password on each side
  - cryptography
  - software / hardware security
- Using/typing the password:
  - *** vs shoulder surfing
- Transmitting the password
  - (encrypted in some way?) neither necessary nor sufficient…
- Destroying the password (why not)

Stealing Passwords



Attacks Taxonomy

- Guessing
- Snooping / shoulder surfing
- Eavesdropping / sniffing
- Spoofing (fake login page)

Impersonation = masquerading = illegitimate access with correct credentials

How to Measure Password Strength

Threat Models for Password Inherent Strength

Without interception:

- Online guessing, pass or fail.
- Offline password cracking.

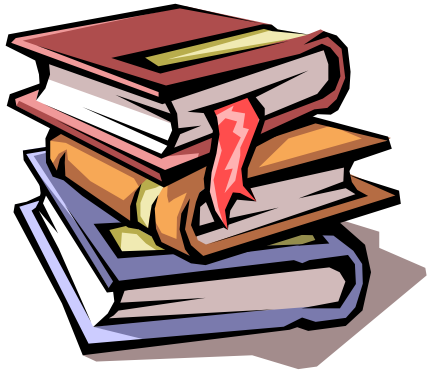
Target:

- against one user
- many users, target one: can be easier!
- target many users

Measures of Strength

- Choosing the password
  - Entropy,
    - single user's password, how hard is it to guess it? A: $2^{3.4}$ attempts.
  - Min-entropy = $-\log_2$(P most frequent password):
    - the weakest == the most frequent password,
    - important in attacks against multiple users
  - Conditional entropy:
    - similar as old password,
    - same as another password,
    - correlated with memorable places dates names etc

Revision About Entropy



A Random Variable

By definition,

a [real-valued] random variable X,

is an application $X: \Omega \rightarrow IR$.

For each realisation of the experiment, X takes some value.

Each random variable has a probability distribution.

Assume that a source X outputs

one of the values $x_1..x_m$.

Then the probability distribution of X is defined by the

p_i =def= $Pr[X= x_i]$.

Entropy of a Source

Again let X be a random variable (with a finite or infinite number of possible outcomes x_i).

The entropy of X [Shannon] is:

$$H(X) \quad =def= \quad - \Sigma_x \; Pr[X=x] \log_2 Pr[X=x]$$

It depends on the probability distribution and :

$$H(X) = \quad - \Sigma_i \; p_i \log_2 p_i$$

*Properties of the Entropy

Joint source:

- H(X,Y) >= H(X) with equality if and only if Y can be written as f(X).

(The joint entropy is bigger than of one source, except if the second source is fully dependent on the first, then Y does not bring any additional uncertainty.)

- H(X,Y) <= H(X) + H(Y) with equality if and only if X and Y are independent.

(When the variables are independent, the uncertainties add up. If not, the uncertainty will be less than the sum of the two.)

Properties of the Entropy

Very important Theorem:

- If there are n possible values x_i with $Pr[X=x_i]>0$, then $H(X) <= \log_2 (n)$ with equality if and only if the distribution is uniform.

(Biased sources yield less information ! (e.g. advertisements on TV). Not much uncertainty in what they will say.)

Corollary: Theorem 12-1 in Bishop

The average expected time to guess a password

[for one fixed user] is maximised when

all the possible passwords are equiprobable.

Proof: from last page:

$H(X) <= \log_2 (n)$ with equality if and only if the distribution is uniform

Conditional Entropy

The same, but the universe "shrinks".

The entropy of X knowing Y

$$H(X \mid Y)$$

It measures the amount of uncertainty remaining about X when Y has been observed and is known.

Conditional Entropy - Formulas

The entropy of X knowing Y (also called equivocation of Y about X):

$$H(X \mid Y) =$$

$$\Sigma_y \, p(y) * H(X \mid Y=y) =$$

$$- \Sigma_x \, Pr[X=x \mid Y] \log_2 Pr[X=x \mid Y] =$$

$$- \Sigma_{xy} \, p(x,y) * \log_2 p(x|y) =$$

$$- \Sigma_{xy} \, p(x|y)*p(y) * \log_2 p(x|y)$$

Measures the amount of uncertainty remaining about X when Y has been observed and is known.

*Conditional Entropy - Properties

- $H(X \mid Y) >= 0$ and $H(X \mid X) = 0$.
  (There is no uncertainty left about X when we know X.)
- $H(X \mid Y) = H(X,Y) - H(Y)$
  (The conditional entropy is equal to joint entropy where we remove the entropy of Y, because we know Y.)
- $H(X \mid Y) <= H(X)$ with equality if and only if X and Y are independent.
  (The entropy of X can only decrease when we know Y. If it doesn't, means that X does not depend at all on Y.)

Mutual Information

- I(X,Y) =def= H(X | Y)-H(X)=H(X,Y)-H(Y)-H(X)
  (how much information is common, symmetric value)

Password Management

Bad User

Users fail to manage passwords properly.

And in various ways.

including highly comical ones.

Bad User?

Users have the right to be bad.

Rules

UCL and many other organizations:

1. Limit the size of passwords. 8 characters max.
   - Why? No reason.
     - so that users would not write them down? They must do it as well now.
     - User's would not forget them? They do anyway.
     - 8 characters are still crackable by brute force.
2. A the same time they make them impossible to remember
   - (must use numbers, &"=+- etc – forcing this can lead to lower entropy if password is short..).
3. Frequent change every few months. Does it make sense?
   - Yes, if we assume that people do sometimes share a password with a colleague (they do, more frequent that any real attack).
   - Makes it even harder to manage, and remember, helpdesk workload.

These choices are not exact science, they are a matter of opinion.

Or ideology. Or misplaced priorities. "Self-Defeating Security"

Chosen? System-Generated?

99% of systems
including most banks
allow people to chose their passwords.

This makes them much less secure.

- lower entropy in general
- even for security-aware users, humans are just TOTALLY UNABLE to generate really random numbers, entropy is just lower!

*Recall: Theorem 12-1 in Bishop

The average expected time to guess a password

[for one fixed user] is maximised when

all the possible passwords are equiprobable.

Proof: $H(X) <= \log_2 (n)$

with equality if and only if the distribution is uniform

Random and Uniform

The most secure policy is random + uniform

- must be system-generated then,
  - very few people can do random and uniform
- incidentally, it is random uniform, it is WRONG to ban PIN=1234 or Password=Alice1234
  - Some people in the UK should have bank card PIN 1234.
    - Otherwise it is easier to guess PINs for other people,
      - lower entropy, see previous Thm!

System-Generated!

The most secure policy is system-generated and truly random.

- Only then it can be relatively short (say 8-12 characters).
- Excellent protection against reuse.
  - But not perfect, this password can be reused in another system that does allow to chose passwords,
    - now if every system generates passwords itself, reuse is impossible.
  - checking for similarity with previous passwords is actually ineffective
    - what? yes, system cannot check with other systems (password reuse)

*Yet Another Method Against Reuse…

Suggested by Yvo Desmedt, privately in 2010.

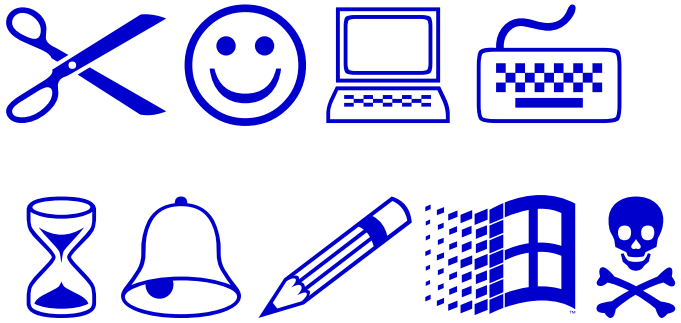
- Can be added to most password policies.
- **Always publish an UNSALTED hash of your password.**
  - Drawback: massive offline cracking will find ALL weak passwords. So works only IF passwords are BOTH
    - unique and
    - very very strong. [machine gen + long enough like 16 chars].

Specific Controls

Most people implement most of these nowadays:

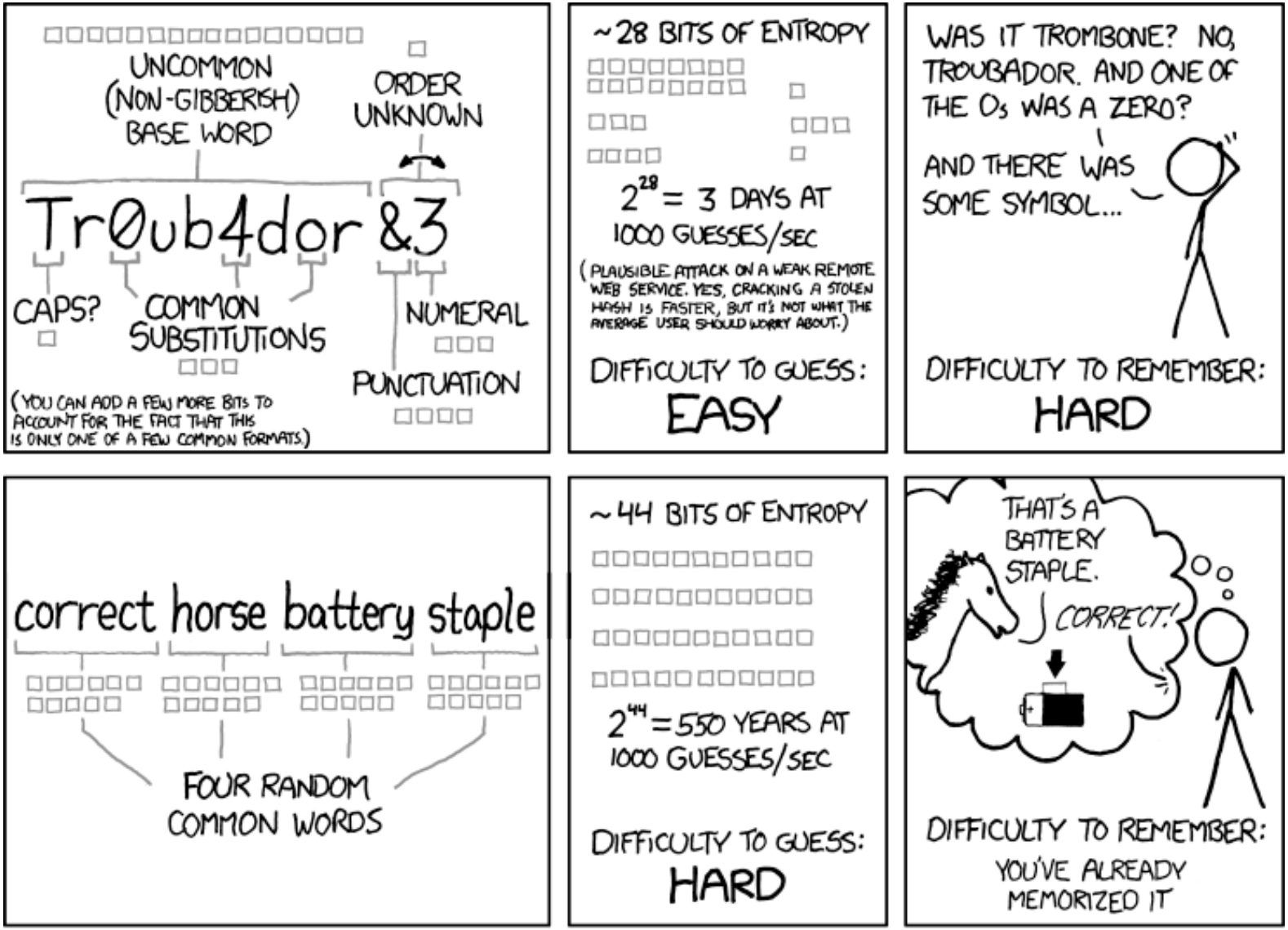
- Compare to last 5 passwords
  - how to cheat: change password 5 times today to erase the history
    - conclusion: keep the whole history
- check if the password is not in a dictionary
  - prevents dictionary attack?
    - not a real one or a more general one,
      -
- maximum validity time policy
  - but to avoid users bypassing that,
    - keep history of recent passwords? Problems, see later…

Human Side



Human-Friendly

- just use very long passwords, but in English,
  - easier to remember at same hardness level?
- Visual Passwords, Pass Faces,
- Etc.



THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Write Passwords Down?

Recommended by Schneier in 2008

Inevitable, people have 1000s of passwords
TIP: DO IT ON PAPER ONLY.

Better than reuse(!). Very practical.

A. Improvement: Write them in an "enigmatic" way, or write a half of it…

- close to 2-factor authentication: Like 1.4 factors.
- sth we know + sth we have all the time… Q: Why it is weak?

Further development: exists commercially but rarely used:

- use a hash function with a secret k, pwd=H(k,input)
- store the input only

Remove Asterisks?

Recommended by Schneier even more recently [2009]

Show the password on the screen…

I strongly disagree with this recommendation.

Again, one can have the best of both worlds.

Example: on old Nokia phones:

- shows each character for ½ second, then replace by a *.
  - nice compromise between security and usability.

Dead or Alive?

Already 20 years ago… passwords were no longer secure OR no longer usable.

- It is not even feasible to remember one really strong password.
  - assuming available data allows offline brute force attacks
    - » hackers can crack 99.99% of passwords,
    - » my students have always managed to crack MORE [LinkedIn Bitcoin etc]

But passwords are very widely used

- new web systems every day…

Secondary Passwords

Like

-Maiden Mother Name

-First School You Attended

-etc

Two problems:

- usually less secure, backdoor entry point
  - [cf. Schneier blog entry]
- legitimate users fail to pass
  - most these questions such as the “name of your first pet” do or “your fist car” do not have a unique answer,
    - problems with spelling, capital letters etc.

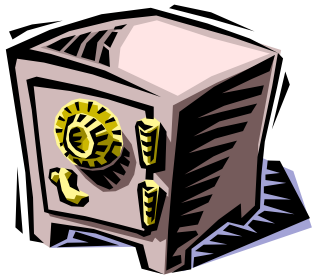
Secondary Passwords

Recommendations:

- If we are good at managing our business [trusting yourself, good memory, accurate records] then
  a good idea is to give a false mother maiden's name,
  - Schneier recommended that people give totally random answers
  - but then you must be able to remember what it was!!! (write it down).

Password Storage

theory / methods



Password Storage and Verification

How to store a password p ?

Method 1: store p. VERY BAD !

Unnecessary point of failure.

Not needed!

Key concept: OWF. One Way Function.

Password Storage and Verification

Method 2: store h(p).

Better but…

Brute force attacks possible though h is a OWF.

Guessing and checking the passwords.

A OWF does protect strong passwords, but it reveals weak ones.

Password Storage and Verification

Method 2: store h(p).

Problem: in a system with many users:

- Example: in very old versions Unix /etc/passwd stored all hashes.
  - world readable
- now for each single password tried,
  - each hash can be tried against all users
    - the attack time can be divided by as much as the number of users
    - h(p) for all users an be stored in a hash table, constant access time -- check for ALL users in O(1).
- similarly, a dictionary attack will be facilitated too:
  - for a $\in$ dictionary and in increasing frequency order, compute h(a) and check if in hash table.

Password Storage and Verification

Method 3:

Key idea:

use a different hash function for each user (!).

- Example 1: store h(name, p).
- Example 2: store h(salt, p), salt.
  - With salt being a random "Shadow ID" for this user.
  - A modified DES-based variant is used in Unix…
  - ***later /etc/shadow is used, readable only by its owner=root
  - now cannot relate password from different users,
  - removes the faster dictionary attack form the last slide

Requirement: OWF. SPR not needed !

Salting

– Method 3.2. h(salt, p), salt.

- with salt being a random and anonymous “Shadow ID” for this user.

Requirements:

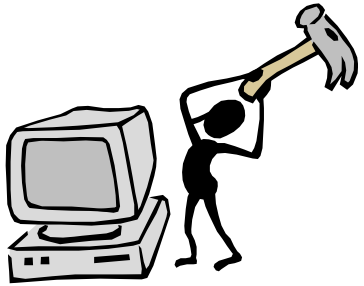
- Should be different for each user.
- Should be also different for the same user in another computer system.

So It Is Better To Have:

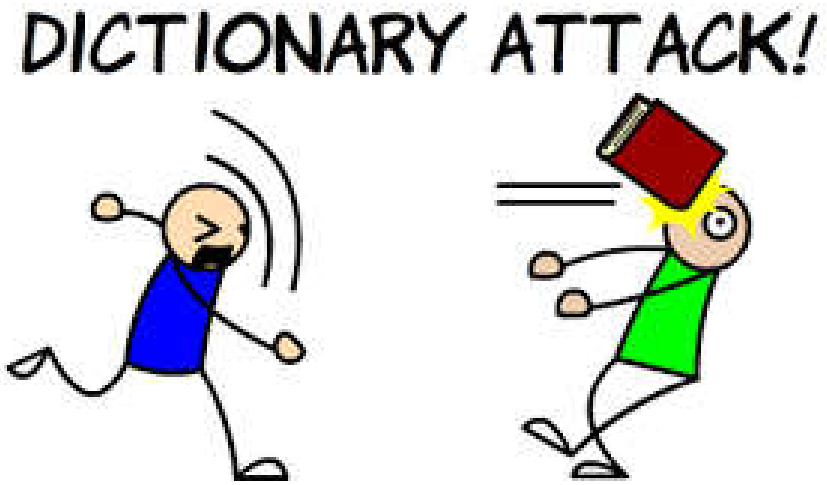
Method 3.3.:

store h(name, machine ID, salt, password), salt.

Hacker Side



- a dictionary attack?
- looking for word78 …
  - no need to store the whole dictionary of English either, some
  - most words have very low frequency and are known to very few people…
- a modern “dictionary attack” need to contain knowledge about the probability distribution of real-life passwords,
  - able to sample this distribution in the order of decreasing frequency



Password Crackers

Most are based on the combination of

- Dictionaries. Databases of frequently used passwords
  - [sample in decreasing probability order]
- time-memory-data tradeoffs, [Martin Hellman 1980]
  - usually implemented using rainbow tables [Philippe Oechslin 2003]

Unix Passwords

Old example:

John the Ripper software:

- claims legitimate purpose,
  - detecting weak password by system admin
  - supports 11 versions of Unix
- cracks
  - several DES-based versions
  - MD5-based
  - Blowfish-based [OpenBSD]

Windows Passwords

Historically quite easy to crack…

Example: In Windows XP: Two methods will work:

1. Boot the machine using OphCrack CD. It will break and show the passwords for all the accounts including admin.

2. Using a bootable live CD OS, replace the SAM file in C:\Windows\System32\config\SAM taken from another machine for which the Administrator password is known. Now one can boot this system using the password from the other system.

Ms Office Passwords

For very long time tools bought on the Internet… 50$.

ZIP and PGP passwords:

harder, tools running on many PCs, 5000$...

How Bad Can This Get? [2012]

In June 2012 a file containing over six million password hashes which allegedly originated from LinkedIn was widely circulated over the Internet.

- Hashes were NOT salted.
- Later hackers found out lots of passwords using rainbow tables and dictionary attacks.
  - Many cracked passwords contained "linked" or even "linkedin" in the form, for example "lawrencelinkedin".
  - Even passwords such as "parikh093760239", "a06v1203n08" and "376417miata?“ has already been cracked…
    - BTW. My students have cracked many more…

Bitcoin BrainWallets:

Private key=
SHA_1(password)

1. Ryan's Castelluci Defcon 23rd

- More than 18,000 passwords were found
- Brainwallet.org closed

2. FC 2016 Bonneau paper [PhD thesis about password cracking, NSA price]

- Median time(money staying in a brain wallet) is < 1 day
- Since Sep 2013 it becomes measured in minutes and seconds
- they identified and traced 14 "drainers"

3. Our Paper

*OFFLINE – Bitcoin BlockChain is public!

Speed Optimizations in Bitcoin Key Recovery Attacks

Nicolas Courtois
University College London
n.courtois@ucl.ac.uk

Guangyan Song
University College London
g.song@cs.ucl.ac.uk

Ryan Castellucci
White Ops
pubs@ryanc.org

ABSTRACT

In this paper we study and give the first detailed benchmarks on existing implementations of the secp256k1 elliptic curve used by at least hundreds of thousands of users in Bitcoin and other cryptocurrencies. Our implementation improves the state of the art by a factor of 2.5, with focus on the cases where side channel attacks are not a concern and a large quantity of RAM is available. As a result, we are able to scan the Bitcoin blockchain for weak keys faster than any previous implementation. We also give some examples of passwords which have we have cracked, showing that brain wallets are not secure in practice even for quite complex passwords.

Keywords

Bitcoin, Elliptic Curve Cryptography, Crypto Currency, Brain Wallet

Everyone on the network can verify the signature that has been sent out. Anyone can spend all the bitcoin in a bitcoin address as long as they hold the cosponsoring private key. Once the private is lost, the bitcoin network will not recognize any other evidence of ownership.

Bitcoin uses digital signature protect the ownership bitcoin and private key is the only evidence of owning bitcoin. Thus it is very important to look at the technical details of the digital signature scheme used in bitcoin.

1.1 Structure of the paper

In this paper we study and give the first detailed benchmarks on existing secp256k1 elliptic curve implementations used in Bitcoin. Section 2 introduces background knowledge about elliptic curve cryptography and brain wallets. Section 3 reviews previous research work in this area. Section 4 gives detailed benchmark for existing method and our own implementation. Our implementation improves the state of the

Table 7.5: Time cost for different window width w for EC key generation

	w=4	w=8	w=12	w=16	w=20
d	64	32	22	16	13
number of additions	63	31	21	15	12
precomputation memory	81.92 KB	655.36 KB	7.21 MB	83.89 MB	1.09 GB
secp256k1_gej_add_ge	45.85 us	22.16 us	15.35 us	11.23 us	9.23 us
secp256k1_gej_add_ge_var	**37.37 us***	17.86 us	12.21 us	8.89 us	**7.16 us**
7M + 4S code	39.01 us	18.79 us	12.77 us	9.23 us	7.48 us
covert Jacobian to Affine	≈ 10 us				
Benchmark on my laptop i7-3520m 2.9 GHz CPU	≈ 42 K guesses / sec (single thread)				
Defcon Attack** i7-2600 3.2 GHz CPU	≈ 130 K guesses / sec				
Improved Defcon attack**	≈ 315 K guesses / sec				

$50 to check a trillion passphrases

* Defcon attack [Cas] is equivalent to this results

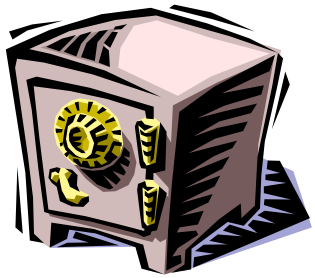
** Results are reported by Ryan Castellucci running his Defcon code and our improved code on 8 threads with linux gcc compiler.

*Some Bitcoin Brain Wallet Passwords

- “say hello to my little friend”
- “to be or not to be”
- “andreas antonopoulos”
- “mychemicalromance9”
- “yohohoandabottleofrum”
- “dudewheresmycar”
- “youaremysunshinemyonlysunshine”
- “THIS IS IT”
- “Arnold Schwarzenegger"
- “nothing ventured nothing gained”
- …

Password Storage

OS Side, in practice

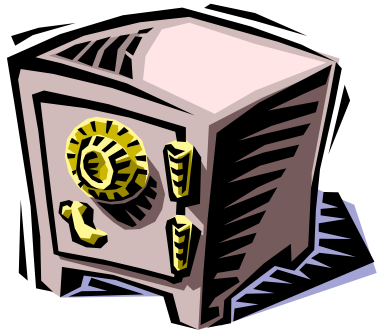


Password Storage on Human User Side

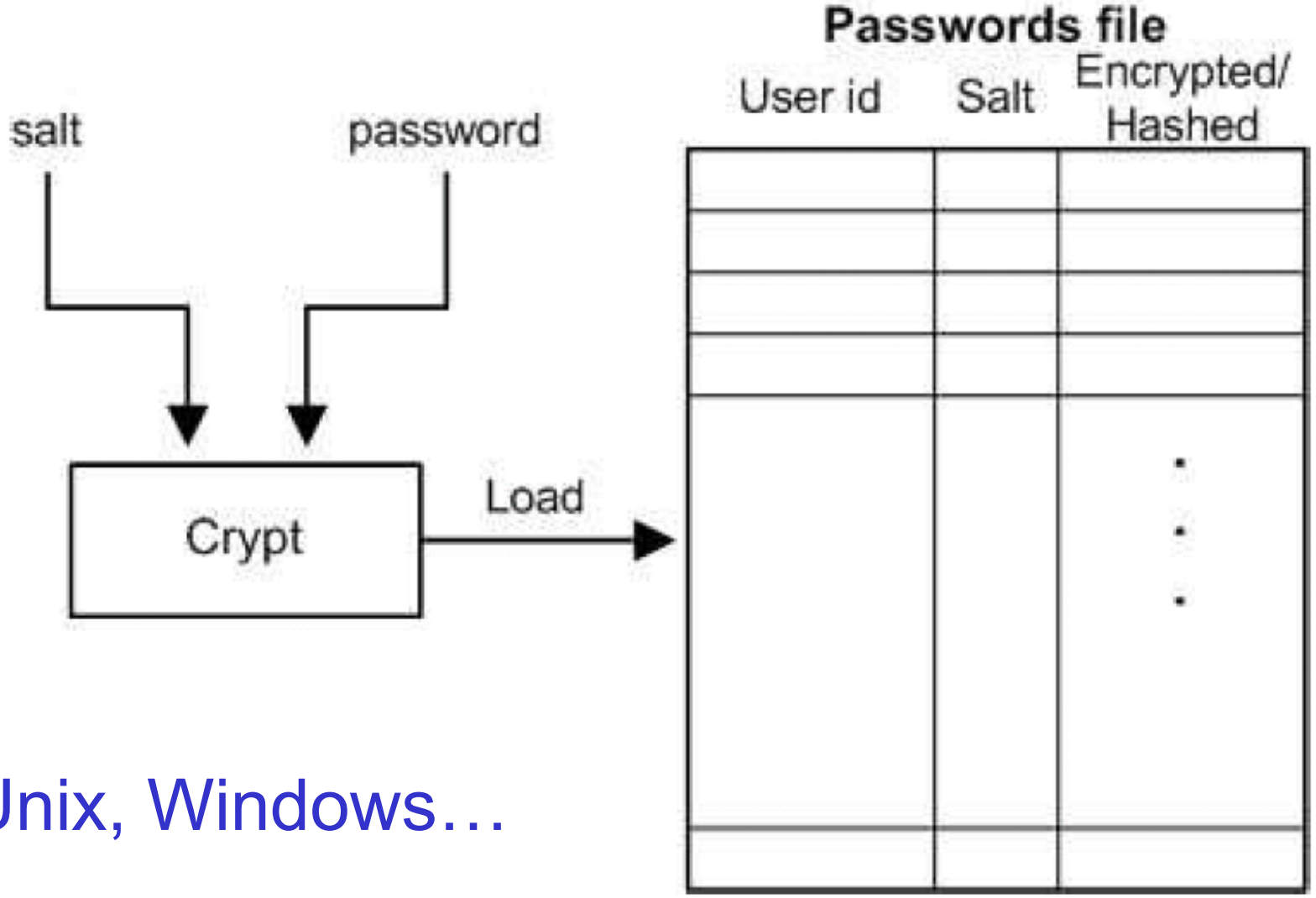
It becomes totally indispensable to keep a log (backup) of all passwords.

Good practices:

- divide in 2-3 categories,
  - financial passwords either
    - complete passwords on paper only and in a safe
    - mnemonics/hint on paper only
  - keep passwords of less importance in an encrypted file, or on paper, or in a mobile phone directory
    - many phones allow to specify which directory entries are stored in a SIM card,
      - » extra PIN protection if phone lost/stolen
      - » but data can still be stolen in real time by a hacker/forensic tool



3.2. – The Most Popular Version



Unix, Windows…

Where is the File?

Unix: /etc/shadow

Windows: C:\Windows\System32\config\SAM

Q: What's wrong with this method?

Windows Passwords

Windows XP:

2. Using a bootable live CD OS, replace the SAM file in C:\Windows\System32\config\SAM taken from another machine for which the Administrator password is known.

- Now one can boot this system using the password from the other system. Better: use chntpw tool, Lab4. Can change pwd, reset, disable, unlock, promote to admin.

Password Storage and Verification

Method 4:

Use a key-dependent hash function (or a block cipher such as AES) and a tamper resistant module that will check passwords.

Hardware Security Module (HSM)

+may add an administrative password or key escrow system to prevent data loss

*Password Storage vs. Size/Entropy

Assuming

best machine-dependent salted hashes == Method 3.3.

Is say 60 bits password secure enough?

Equiv. 12 characters like “_gme1&ls&:”

Is there a “birthday paradox” attack in 2^{30} ???

*Password Storage vs. Size/Entropy

Assuming
best machine-dependent salted hashes == Method 3.3.

Is say 60 bits password secure enough?
Equiv. 12 characters like “_gme1&ls&:”

No birthday paradox attack.

60 bits practically secure, but 80 bits is better.

Nice trick: use a slower hash function to slow down the attacker. This allows to have more human-friendly (shorter) passwords like 50 bits.

**Not Too Similar To Any Older?

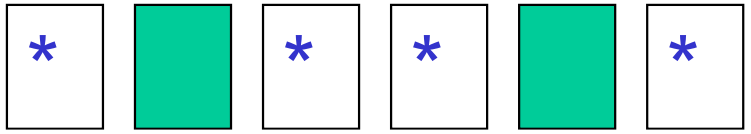
Last case:

- check if not too similar to several older passwords?
  - Very hard to implement this correctly
    - so IT people that imposes this policy made our systems less secure, including at UCL
  - best implementation:
    - store older passwords in cleartext
    - and the current password hashed cf Method 3.3,
    - enforce system-generated passwords

Defences

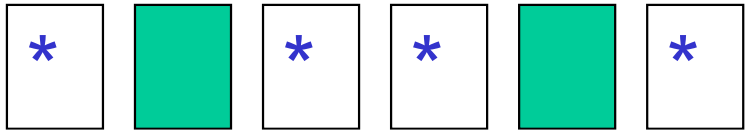
Limited Disclosure Schemes

Used by many banks,
please type digits 1,3,4 and the last.



Limited Disclosure Schemes

Used by many banks,
please type digits 1,3,4 and the last.

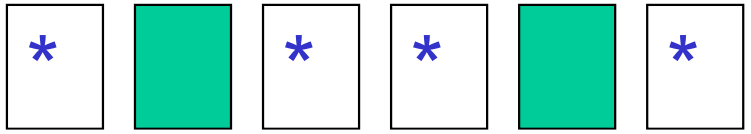


There are non-trivial implementation choices:

- Q1: How many wrong attempts before system locked say for 1 day?
- Q2: How many wrong attempts before system locked forever?

Q3: Same Or Different Subset?

Used by many banks,
please type digits 1,3,4 and the last.



- Next login (after success): use different subset
- Next login (after failure): ask for THE SAME subset, better
  - resist better to shoulder surfing.
    - If the user is careless or has bad memory, repeated re-authentication increases the exposure – more data is leaking, and shoulder surfing can compromise the whole password instantly, this can be mitigated by time delays between attempts.

- Attack: Spoofed login screen will claim it was wrong and ask for a different subset 3 times, enough to capture everything.

Limited Disclosure Schemes

How to store these passwords?

Store individual characters???

NOT GOOD, BRUTE FORCE each character at a time!!!

No good solution, [very large storage would be needed].

Possible solution: hash some but not all subsets of 7 out of 24.

Does not work well, entropy of each cases too low against brute force/dictionary… Advanced attacks: cf. FC'13.

Defences Against Attack Attempts

And against legitimate users that are very frequently are unable to log themselves…

Main policies used in practice:

- Disconnection:
  - just release the connection for now, no sanction
- Disable the account after several unsuccessful trials
  - denial of service, can be used in your office but not on the Internet
- Backoff:
  - add a delay at each attempt,
    - typically grows exponentially, but can be slow, e.g. $(1.1)^n$
- Jailing:
  - if password is incorrect, give the user access to a fake system…
    - could be used more frequently but it is very hard to produce a false systems that looks genuine, so this usually just allows to follow a few first steps of the attack, seeing what he is up to or what he is primarily looking for

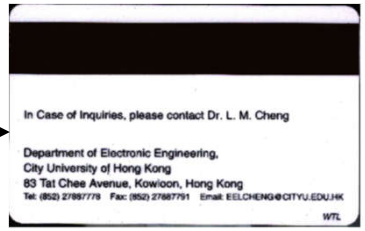
Silent Alarm Password/PIN

Defense against an attacker
that forces us to reveal the password / PIN.

A secondary password / PIN that is accepted
but raises an alarm. A.k.a. “duress PIN”.

Passwords = Static Authentication

Skimming Bank Cards



Can We Do Better?

Beyond Passwords

In the real world, passwords are

- low entropy,
- yet impossible to remember,
- shared,
- reused

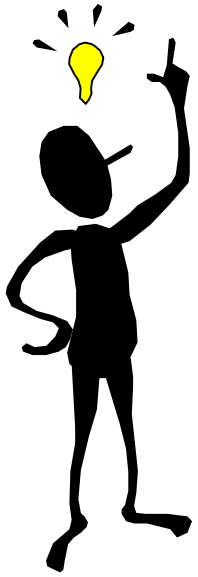
Hackers do

- guess / crack them
- intercept/record and replay

Can we defend against all these?

- reset passwords frequently… check if strong
- or move from static to dynamic schemes!

Dynamic Authentication



Dynamic vs. Static Authentication

Dynamic as opposed to static.

dynamic (authentication) systems:

- One-Time Passwords (OTP),
  - in any order, counter-based, frame number-based
- time based
- random challenge-based
- data-based
- data+challenge based

One Time and Better…

One-Time Passwords (OTP)

Key properties:

- The password is changed each time
- The attacker cannot know it in advance,
  - real-time MIM attacks remain possible
- The fraudulent authentication attempts are detected

Lamport OTP Scheme

Based on OWF.

Use hash chains.

Go backwards.

Let $x_1=h(x)$, $x_2=h(x_1)$, …, $x_{1000}=h(x_{999})$.

Store x_{1000} on the server. Small storage. Fast.

Go backwards, reveal x_{1000} at first attempt.

Then reveal x_{999} etc.

Each x_i allows to log-in only once.

One-Time Passwords in Banking

A card with printed random numbers.

Problem: can be photocopied…

- and the user still has it, naively thinking it is secure…

One-Time Passwords?

Time-synchronized OTP

RSA SecureID etc.

Wrong:

This is NOT a OTP scheme.

Misnomer = “OTP token” ????

Not really a OTP,

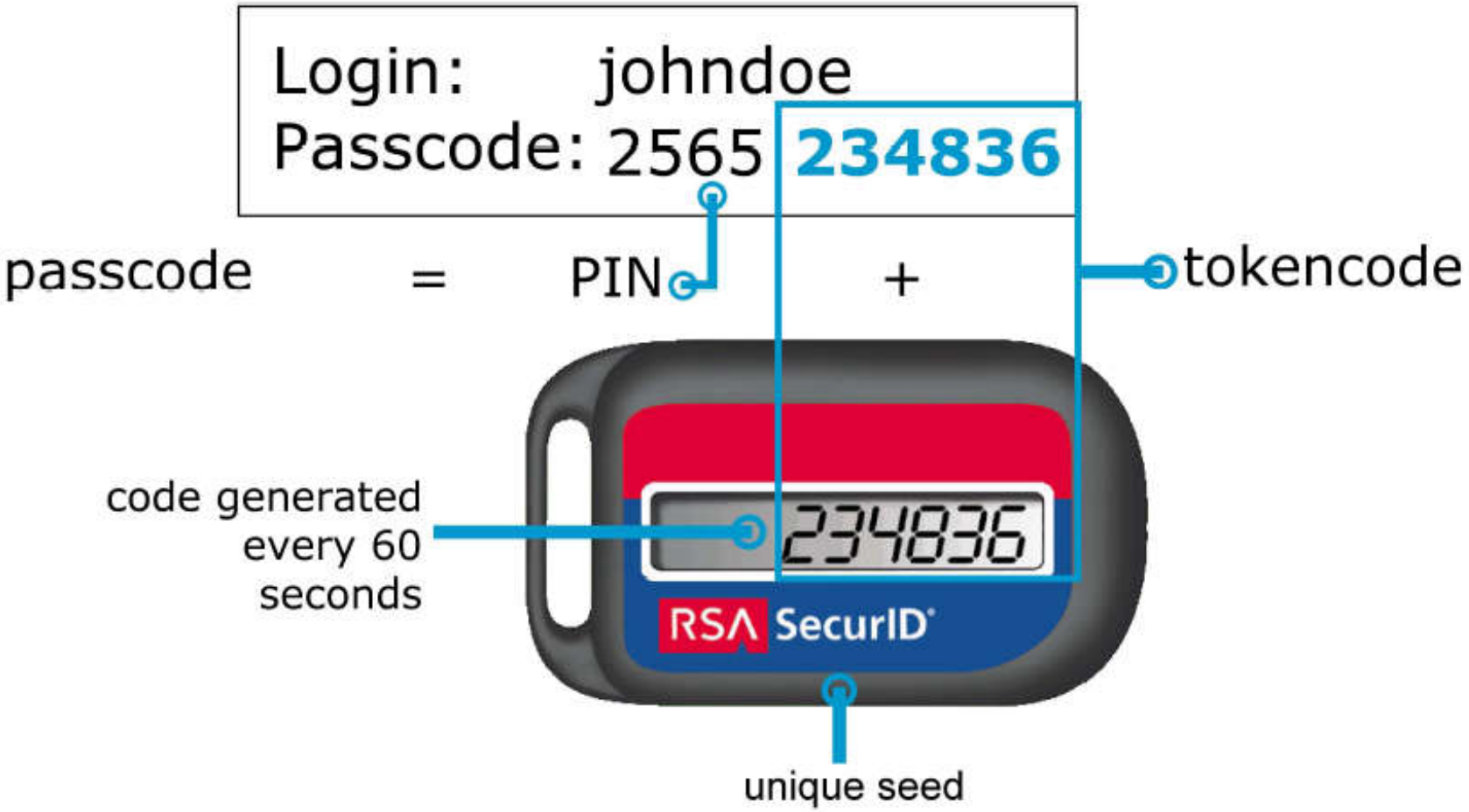
it is nearly a challenge-response system, kind of half way.

Where challenge = time.

- Except that the challenge is fixed for 30-60 s.

  Window of opportunity: 30 s, second session possible connected from another location…

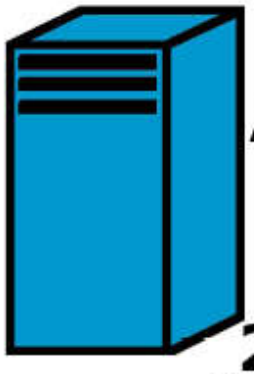
RSA SecureID is a 2-factor System



=> PC login…

Proprietary Symmetric Algorithm

Authenticator



RSA
ACE/Server

234836

Algorithm

Time

Seed

234836

Algorithm

Time

Seed

(secret)

identical seed

identical time

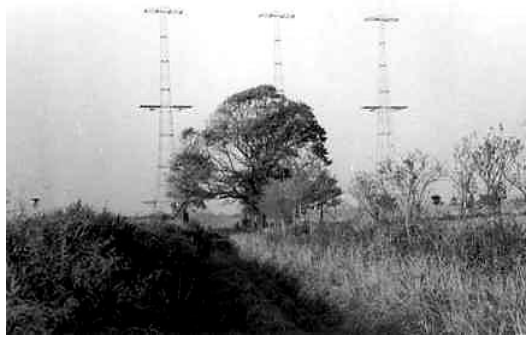
Challenge-Response Protocols

- Better,
  - the right answer to replay attacks.

C-R Authentication - History

IFF: Identify Friend or Foe (1942)

Challenge-



-Response

MIM ≥ Relay Attacks

Mitigated by precise measure of timing.

Nonces and Time Stamps

Nonces

nonce = a number used once, counter/sequence number

» less secure

in the sense of challenge-response

random nonce = a random challenge = a random

Warning: frequently, a random nonce will be called just nonce, but what is meant is a random nonce.

Time vs. Nonce

Time can replace a random nonce, can simplify protocols,

- between very slightly and a lot less secure,
  - mainly depending on time granularity.

Dynamic,

half way between static authentication and challenge-response systems (the best).

Timestamps in Computer Science

A time stamp is a standard way of representing the time.

Protocols use a standard representation.

- ISO 8601 covers all these.
- Examples:

2007-11-09 T 11:20 UTC

Sat Jul 23 02:16:57 2005

1234567890

Unix time: the number of seconds since 00:00:00 UTC on January 1, 1970 UTC

Secure Time Stamps

Timestamps in Cryptology

A secure time stamp = Time Stamp Token = TST

is a token that is

1. unforgeable [e.g. with digital signatures]
   - *can be expensive to insure,
     - CPU cost, timestamp length, PK infrastructure
     - Q: can it be verified in real time?
2. will depend on your data/context
   - if you hash it together with a random number, data remains private
   - the simplest case: this data will be the time itself alone
3. cannot be created before a given time.
   - example: contains the title of todays' newspaper [Satoshi Genesis block 2009].
4. cannot be created later either,
   - publish a hash of your timestamp in a blockchain!!!

Old Centralized Solutions:

Time Stamp Authority = TSA

Time Stamp Token = TST

ANS X9.95 based on:

- IETF 3161 Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)
- ISO 18014 Time Stamping Services …

New Solutions:

Bitcoin and Ethereum blockchains.

+ data storage businesses
    [only hashes need to be stored in blockchains]

Part of large cloud IT business [Amazon Microsoft]
    underpinned by EOS/Ethereum/NEO etc?

Time Stamps and Digital Signatures

If a secure timestamp is included in data signed by the digital signature, it further increases the power of non-repudiation now to guarantee:

- who
- what
- when

Uni-directional Authentication

Unilateral Authentication

statement,

[interactive] proof

Unilateral Authentication

Historically very popular.

Examples:

- password -> login

- SIM card -> GSM base station (fixed in 3G)
- offline bank card transactions -> Point of Sale terminal

Problems:

- login page spoofing etc.
- false GSM base stations,
- false ATMs,

Unilateral with Time/Counters

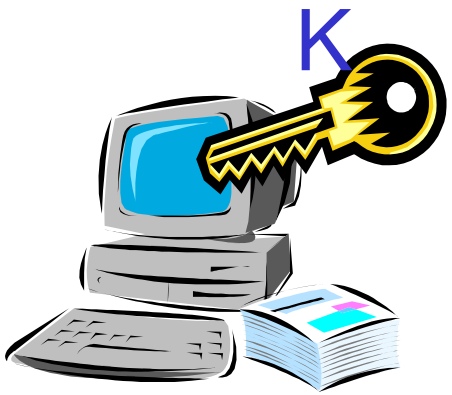
Unilateral Authentication

Let tc be indifferently:

- current time
- a secure (i.e. cryptographic) time stamp
- a nonce in the sense of a counter
- a sequence number

Any other value that is
sort of guaranteed
never to repeat.

Unilateral with Time/Counter



$A, tc, MAC_K(tc, B)$

A

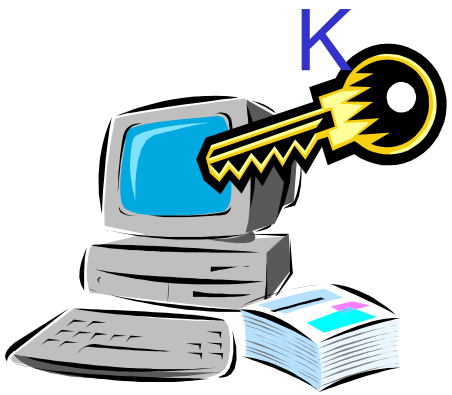
B

can also use a block or stream cipher, but always works as a MAC here

Q1: why the name of A is included?

Q2: why the name of B is included?

Unilateral with Time/Counter



$A, tc, MAC_K(tc, B)$

A

B

can also use a block or stream cipher, but always works as a MAC here

Q1: why the name of A is included?

Q2: why the name of B is included?

Reflection attack: reuse when B authenticating to A concurrently without knowing the key.

Unilateral with Time/Counters - PK Versions

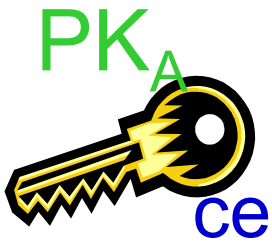
PK-Version with Digital Signatures

$A, tc, S_{SK(A)}(tc, B)$

A

B

Further PKI Version



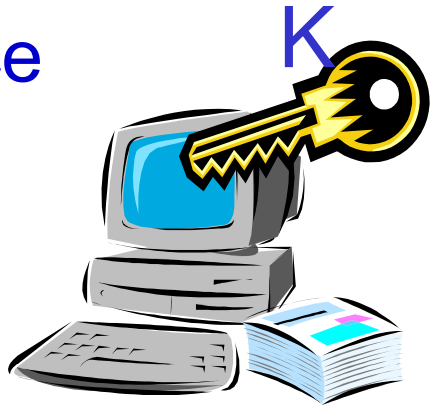
$cert_A$, tc, $S_{SK(A)}$(tc, B)

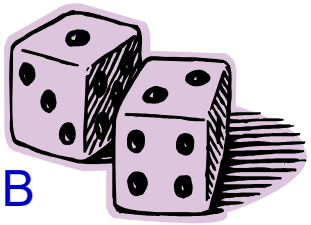
A

B

Unilateral with Random Nonces (better)

Unilateral with a Random Nonce





$random_B$

A

B

$A, MAC_K(random_B, B)$

Unilateral with Random Nonces - PK Versions

Public Key – Based Schemes

Here more possibilities than with time.

Two approaches exist. Alice has her private key SK(A).

Two methods to demonstrate the knowledge of this key:

- sign a message chosen by Bob.
- decrypt a message encrypted by Bob with Alice's public key.

How?

PK-Unilateral with PK Encryption

$h(r_B), B, E_{PK(A)}(r_B, B)$

A ← B

r_B

Q1: why we do have $h(r_B)$ in the first message? A witness: prevents CCA.

Q2: why we do have B twice in the first message?

PK-Unilateral with PK Encryption

$h(r_B), B, E_{PK(A)}(r_B, B)$

A ← B

r_B

Q2: why we do have B twice in the first message?

Vaguely compelling reasons: Guarantees good independence of different sessions. Bob's identity is known and the person that produced the random is the only one that is able to know who B is. Alice checks if B=B before she replies.

Same with PKI

$h(r_B), B, E_{PK(A)}(r_B, B)$

A

B

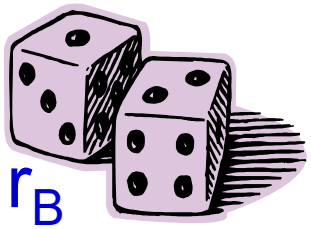
r_B

Public Key – Based Schemes

- decrypt a message encrypted by Bob with Alice's public key.
- sign a message chosen by Bob.

With PKI, second solution is more practical!

PK-Unilateral with Dig. Sign.



A

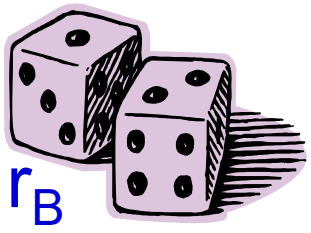
B

$cert_A, r_A, S_{SK(A)}(r_A, r_B, B)$

Q1: why we don't have A here? Unique key.

Q2: why we have added here r_A?

PK-Unilateral with Dig. Sign.



A

B

$cert_A, r_A, S_{SK(A)}(r_A, r_B, B)$

Q1: why we don't have A here? Unique key.

Q2: why we have added here r_A? Again CMA, prevent signing messages entirely chosen by a potential attacker + subtle reasons: allow audit/freshness even if Bob's random numbers repeat.

Quiz

- Can passwords be stored encrypted by a deterministic block cipher algorithm with a fixed key, encryption being implemented in hardware?
  - What if the storage is also implemented in a secure hardware?
- Give two examples of self-defeating security recommendations regarding passwords.
- How to use a hash function to store a password?
- Does it require a OWHF or CRHF?
- In which case the entropy measures the strength of a password?
- What is min-entropy and in which case it measures the strength of a password?
- What is “spoofing” in the context of password security?
  - does “spoofing” require any “sniffing”?
- What are the three factors? Why writing the password down defeats a 2-factor system without necessarily making it less secure? Solutions?
- Which one is better: random challenge-based or time-based authentication?
- Should user-chosen passwords be limited to 8 chars by system admin?
- What is a secure cryptographic timestamp?