Lec 11: web Security

Question: How to isolate sites in a web browser?

Answer: SOP

TM/assumptions: attacker controlled site, you visit their site, Browser is safe

$\boxed{\text{SOP}}$ — script can only access a resource if they have same origin

- prevents attacker.com JS from talking to gmail
- XMLHTTPRequest can only fetch data from same origin

postMsg is non-SO comms

How to change Location of Existing Documents

1) frames

2) target on hrefs

• $\boxed{\text{CORS}}$
- specific HTTP headers on serverside
- like image retrieval, links, scripts, i-frames
  - ↑ But SOP prevents interaction

• $\boxed{\text{XSS}}$ — cross-site scripting
- <script> bad code </script>
- use CSP Http header

How do we know data is really from server? SSL + certs

How does the server know the request is real? Cookies.

$\boxed{\text{Cookies}}$
- always sent browser → server
- $\boxed{\text{CSRF}}$ — attacker.com: <img src="http://bank.com/xfer?amount=500 & to=attacker">
  - uses logged in user w/ valid session key in cookie
  - —solution— attach random token, so bank.com knows its bank.com making the request

* SOP doesn't allow code inside iframe to refer to resources outside iframe

Lec 12: Network Security (TCP/IP)

1) Client $\xrightarrow{ISN_c}$ Server (ISN = init sequence #)

2) Server $\xrightarrow[Ack(ISN_c)]{ISN_s}$ Client

3) Client $\xrightarrow[Ack(ISN_s)]{}$ Server

4) Client $\xrightarrow[Data]{}$ Server

⇒

1) Attacker $\xrightarrow{ISN_a}$ Server (pretends to be Client)

2) Server $\xrightarrow[ISN_s \; Ack]{}$ Client → this will trigger Client reset, but SYN flooding server port on T will cause message to be lost

3) Attacker $\xrightarrow[ISN_s \; Ack]{}$ Server, src = T → Guess ISN by Attacker → server to find current ISN, +64 for next one

4) Attacker $\xrightarrow[Bad \; Data]{}$ Server

Defense → be better

Source Routing Abuse: Attacker pretends to be trusted IP addr

Defense → SSL, filtering out forged IP source addrs, DNSSEC

DoS attack
- Clog server ports by sending a bunch of requests that don't respond (server waits for step #3)
- Defense - stateless server until receives ACK (SYN cookies)
- $ISN_s = SN_c$ + (timestamp || SHA1(src/dst addr/port, secret, timestamp))

server sends to client

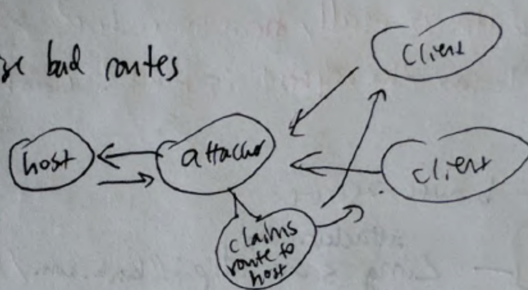
per-client (attacker can't guess)

coarse-grained

hash changes, not useful if stolen

<u>Bandwidth Amp</u> - send ping packets to broadcast addr, fake a victim's address
- Def: can't send packets to broadcast addr

<u>Routing Info Protocol Attacks</u> - advertise bad routes



Sols
- paranoid gateways filter packets on src/dest addr
- no reason to accept new routes on local net
- authentication of RIP packets?
- Trusted DB of announcers
- Combo of link/app layer end to end encryption?

- use of ICMP redirect messages to redirect gateways
- ARP - impersonate router and send a reply for next Ethernet hop
- DHCP - impersonation of server to new clients and choose DNS servers
- BGP - anyone can announce a route
- DNS is a shitshow

Lec 13: Secure Chanels

Question: How do we add authenticity / confidentiality to TCP/IP?

Answer: TLS/SSL

SSL → handshake — key exchange protocol that inits syncs crypt across endpts
record layer — conf / auth / replay protect

SSL2 vs 3
↓
- weakened auth keys to 40 bits
- doesn't authenticate padding

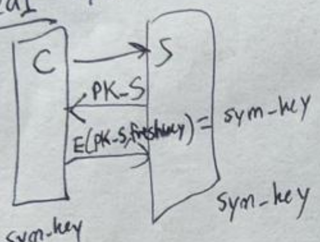
OPERATIONS

- KeyGen()
- Encrypt(public_key, message) → ciphertext
- Decrypt(secret_key, ~~message~~ ciphertext) → ~~cipher~~ message
- Sign(secret_key, message) → signature
- Verify(public_key, message, signature) → ok

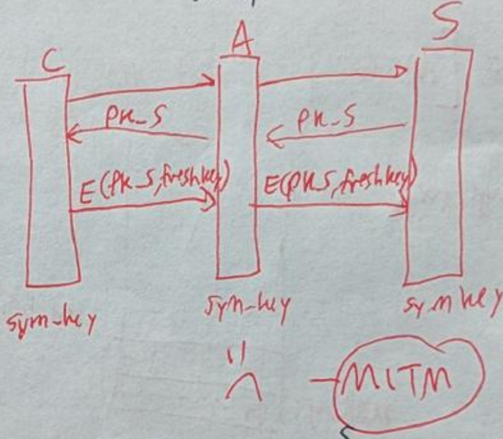
~~MAC~~

Symmetric key ops → use a symmetric key for both encryption & decryption
→ use ~~same~~ diff sym key for MAC (MAC(key, message) → tag)

Idea 1 - symmetric crypto



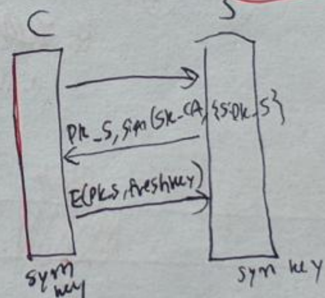
consider



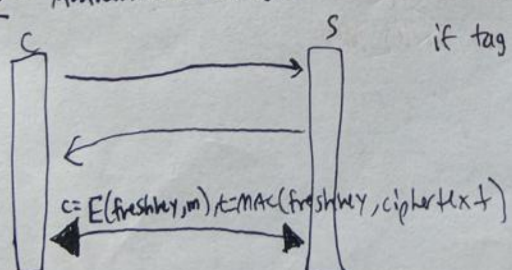
① How do we make sure we're talking to real S?

② How do we make sure messages aren't changed?
- encryption = confidentiality, not integrity

Idea 2 - Certificates
- {principal name, public key} pairs
- Sign(secretkey_certauth, {s. PK_S})



Idea 3 - Authenticate messages



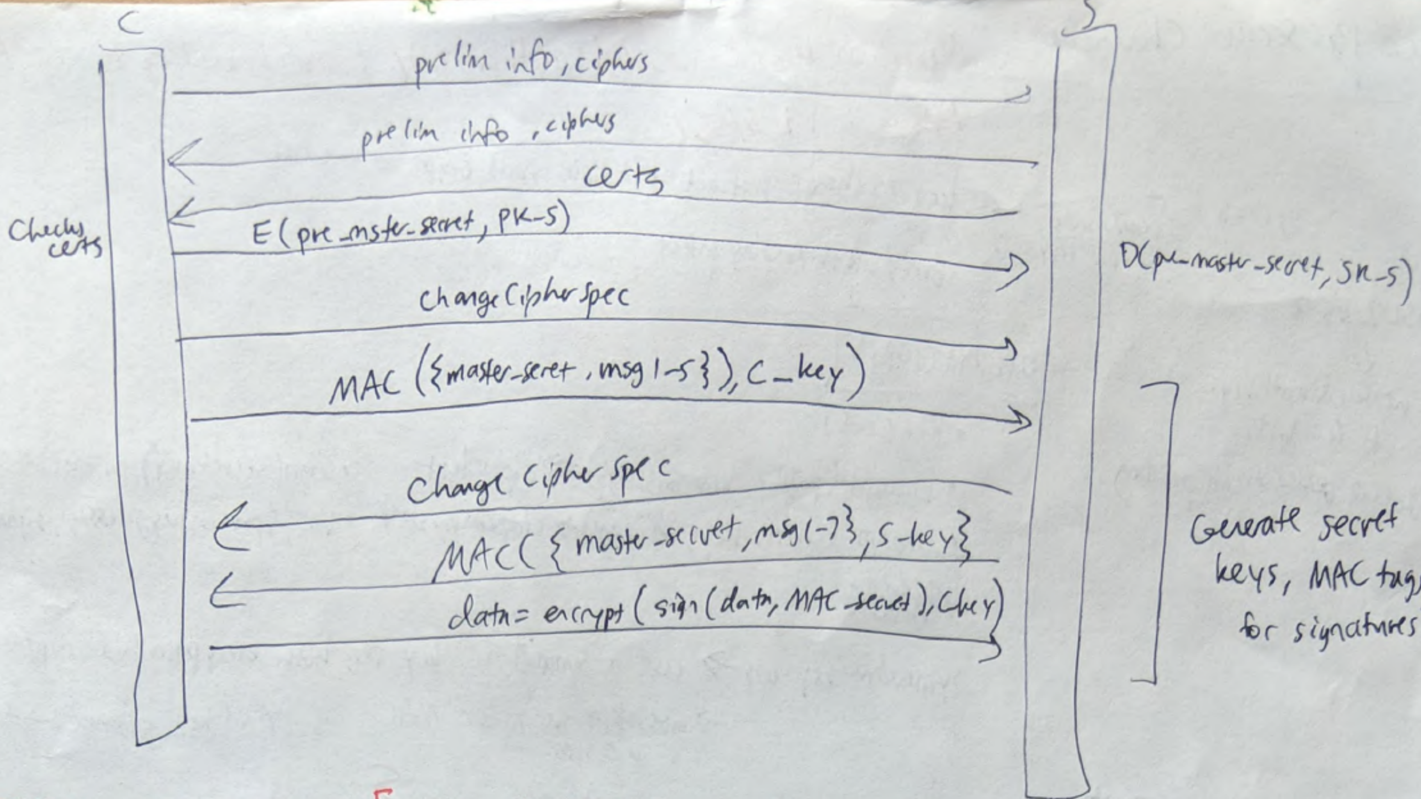
if tag changes, then message has been changed.

① What about replay attacks?
- Sending valid messages 1000x

Solution: - discard already processed msgs
- sequence # in msg
- expiration time

② What if someone compromises the server and can use SK_S to decrypt?

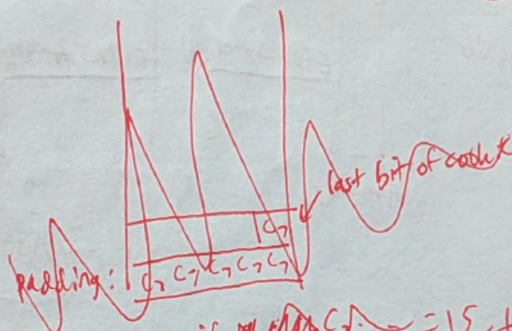
Use short lived keys for encryption, long lived keys for signing (forward secrecy)



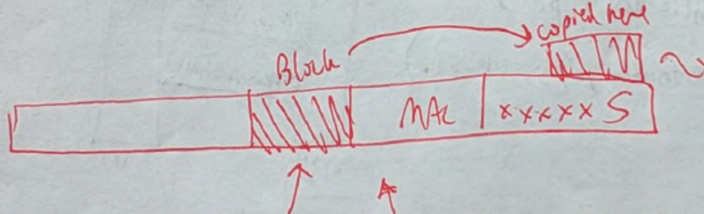
POODLE = SSL message = [MSG | 20-byte MAC | padding] st its %8==0

↓ stealing cookies

Arrange for



if C_7 = 15, then the whole padding block will be removed, and server will accept the msg



if last byte here = right S, then server will accept. Then adversary will learn that last byte is 15 (or whatever).

Client: CBC encrypt this

Server decrypts this, and checks MAC

Lec 14 | Key Distribution via Certs

CA - organization trusted to validate DNS name

<u>Checks</u>

- ✓ subject name in cert = URL name
- ✓ CA is recognized
- ✓ CA signed cert
- ✓ cert is not expired, or revoked
- ✓ server knows private key corresponding to cert public key

If attacker.com shows real.com cert — you have encryption!

If attacker.com shows attacker.com cert — attacker.com != real.com

acme.com
ACME PK
Expiration
issuer ID (CA)
CA sig

cert

EV certs - only legit owners

DV certs - technical check for domain ownership

- CA: "Can you put a nonce into the file?"

Requester

CA → reads checks

* only server owner could create such a file

* depends on DNS/IP

Bogus key changes? Like from sloppy CA's...

Sol: key pinning (Trust on First Use) by browser history. Avoids certificate substitution

Serverside key pinning - avoids certificate substitution and lets server decide what to keep the same or when to change

Lec 15 - Side Channel Attacks

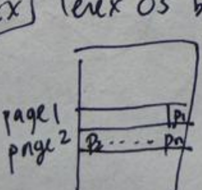
load speed, image size, server response speed

Speculative Execution - running code ahead of execution via branch prediction

- brings vals into cache
- on failed spec, data remains in cache

Ex Tenex OS bug

measure error time - if long, then you know p_1 matches and OS had to load in page #2.



adversary aligns

1) You need to train branch predictor
2) must be able to access arr2
3) must EVICT sz, array2 from address space

Attack

```
if(offset < sz){
    v=arr1[offset]
    v1=arr2[v]
}
```

attacker can't read "v" but...

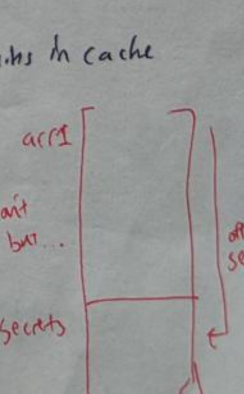
BUT arr2[v] is still in cache

↓

Attacker tests arr2[0] (slow)

arr2[1] (slow)...

arr2[13] FAST! v=13.



arr2 - not in cache

Lec16: AirBNB

Problem: security is not scaling like everything else

Security = Strategy to address risks in your system
=define threats, responses

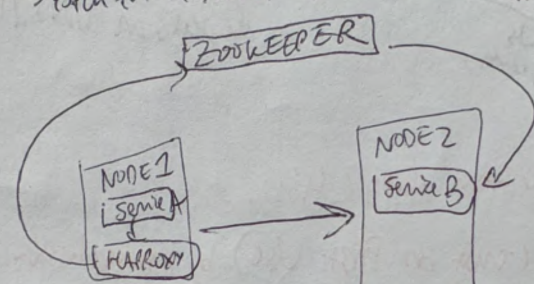
Mitigate risks: threat-agnostic (assume failure)
defenses around pts of failure
self-assess (human trial)

Problem: security focuses on external immediate threats, but ignores internal attacks.

Network segmentation is hard
zoning parts of your network to restrict access (reduced attack surface)

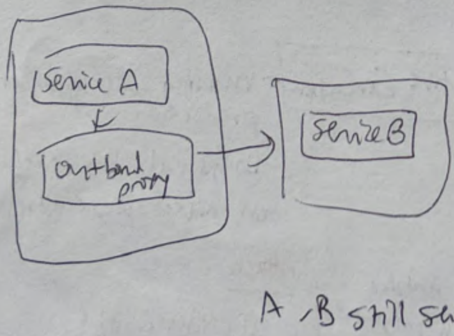
→ SOL: unify security dev & software dev process

* mutual TLS = certs sent both ways (~~client recipient~~ service both send each other)
* service discovery = System for a node to find another node
* SmartStack =

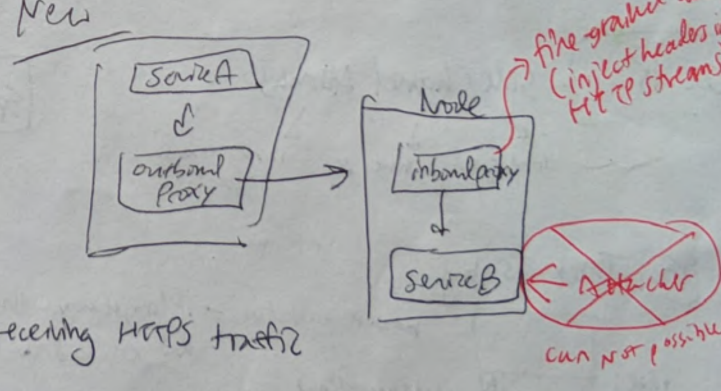


MAP - infer from code
- taking existing config and turning it into a map (Apache)

Old



New



A, B still safely receiving HTTPS traffic

* Segment by SERVICE

* Make IDs varied, unchangeable. Define which IDs can access each node.

Lec 17: Messaging

Prioritizing confidentiality, authenticity

1. TRUST ESTABLISHMENT
2. CONVO SECURITY
3. TRANSPORT PRIVACY

* Hop by hop security

O → O is good

O → O is good

BUT can you trust every node end to end?

does every node have updated security patches?

* End to End: security only depends on security of nodes at end — middle could be bad... and it's OK

Alice → Bob: ciphertext = E(publickeyBob, message)

Alice → Bob: sig = Sign(SecretkeyAlice, message)

Bob: verify(publickeyAlice, c, s)
Decrypt(secretkeyBob, c) = m

Replay Detection? Not really :(

Also what if we want more than one-time communication?

simp. TLS secure channel

(2) Convo security

Alice → Bob: m = AliceBob(randomA#)
E(pkB, m)
Sign(SK-A, m)

Bob → Alice: m = AliceBob(randomB#)
E(PKA, m)
Sign(SK_B, m)

replays after a convo? No b/c random #s

replays during a convo? seq #

$\{K_i,\ i \in [1,n]\}$ = Hash(randomA# randomB# i)

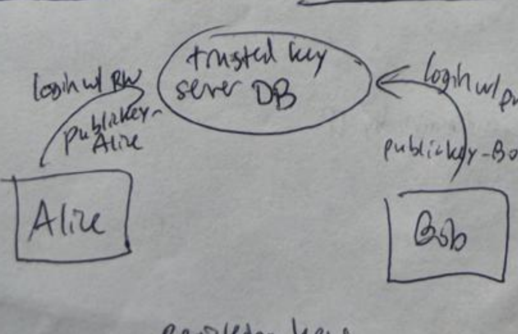
Alice → Encrypt(K_1, m + seq#), MAC(K_2, m) → Bob

(1) Trust Establishment

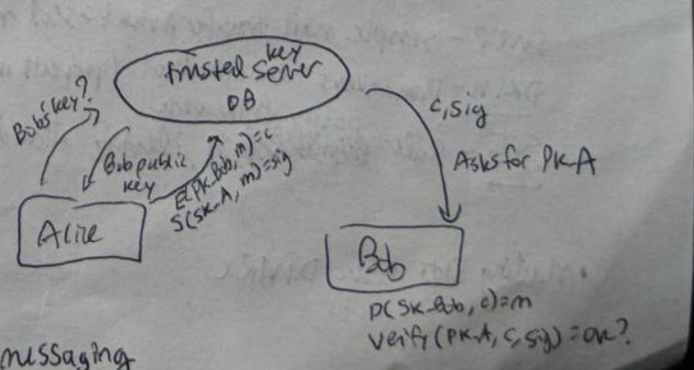
How to find a user's public key?

Public Key Discovery — TOFU assume first cnxn OK and remember this key

Public Key Discovery — Central key server



Registering keys



messaging

How to hold corrupt servers accountable?

SOL: make public log of all key updates! So if server returns Eve's key when Alice asks for Bob's, then server must put "Bob → Pk_Eve" in log. !!

Idea for forward secrecy ~ if Alice has PK_temp / SK_temp today, then attacker can't decrypt stolen packets tomorrow bc SK_temp doesn't exist anywhere anymore.

③ ← Transport Privacy

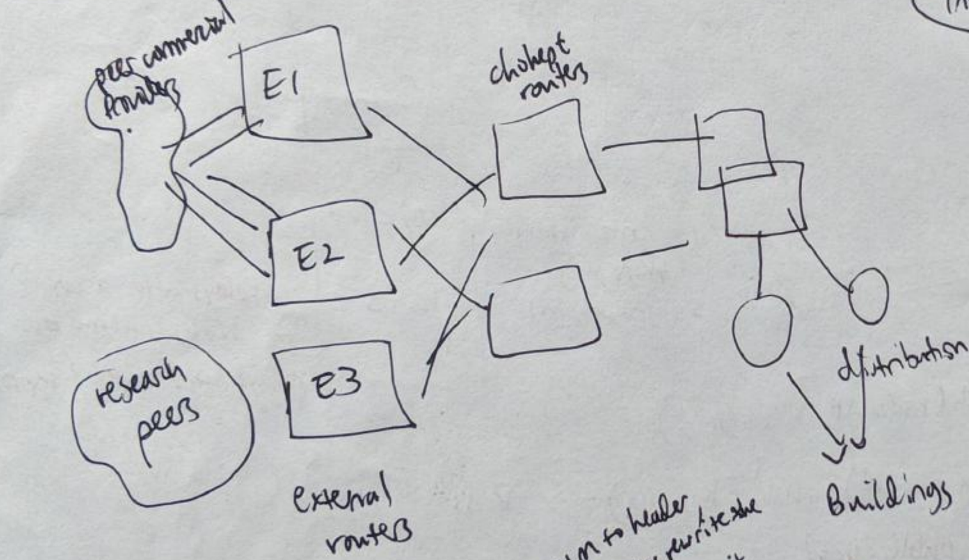
Deniability

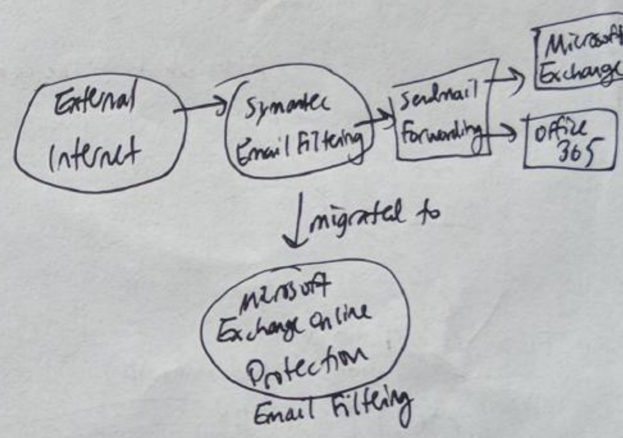
Bob - chooses K (some key). Alice doesn't want anyone to know Bob talked to her.

1) B → E(Pk_Alice, K) → A Anyone could've done E(Pk_Alice, K)

2) A → M → B (MAC(K, M)). A publishes K Anyone who knows M can produce (MAC(K, M))

Lec 18: IS&T





Support for SRS for email forwarding

MTA - mail transfer agent — add DKIM to header but may rewrite the message, so it breaks

SRS - sender rewriting scheme

SMTP - simple mail transfer protocol - send mail → rewrites envelope from to fix SPF

DKIM - DomainKeys Identified Mail (prevents mail spoofing)

SPF - sender policy framework, identify allowed sources of email for a domain by IP

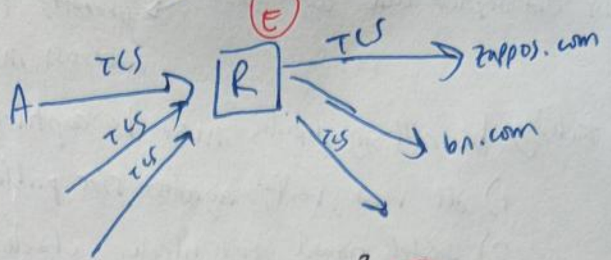
- Mailing lists broke DMARC

Lec 19 Tor

Anonymity. Unlinkability (can't link Alice to any online profiles). Unobservability (can't tell Alice is doing anything at all)

Threat Model — useful for web browsing, interactivity

Building forward anonymity:



what if someone is watching relay?

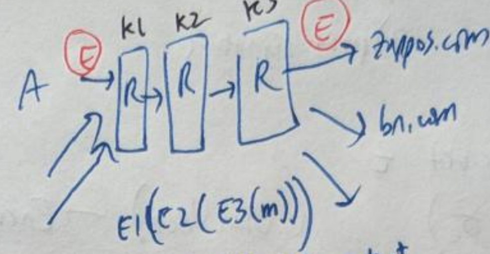
Packet in still = packet out, Eve can still read and determine its Alice on zappos.com

Node Discovery

- if you want to be a relay, authorities vote on you
- informing people on relays

↓

send snips (auth'd by authority) of routing table, esp of relays where they're going. Verifies that routing table honours client's request for random routing



Onion → decrypts at every level. Es prob sym keys negotiated w/ each R and A

Keybase

sol: per-user keys

* MULTI-DEVICE SUPPORT
* Naming teams w/ changing members
* auth'd invitation of new members

• forward, backward secrecy

chains, shared secret key that rotates

Threat — adversaries own server infrastructure, locked devices

team → random shared sym key. rotates when users/devices are revoked.

→ msgs signed by user (auth)

D1

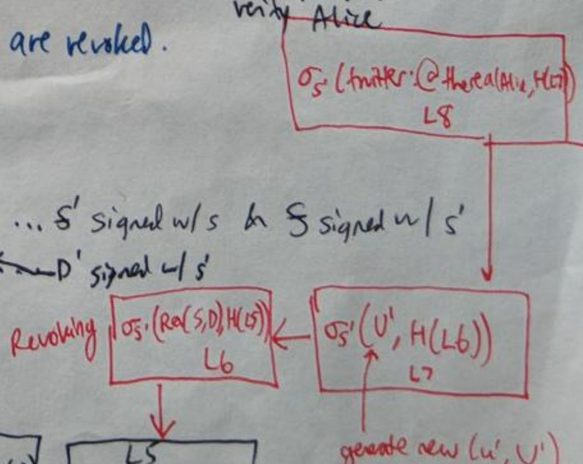
Alice → { username; signing pair: (s, S) ← signed w/ s; DH key pair: (d, D); user DH key pair: (u, U) Encrypted for D, D' }

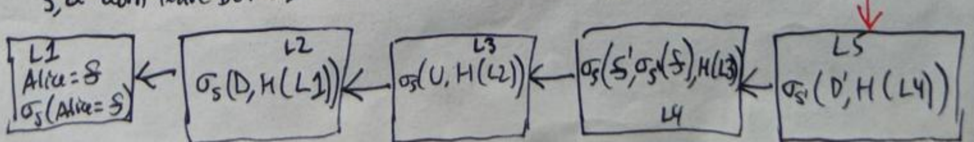
D2

{ (s', S') … S' signed w/ s & S signed w/ s'; (d', D') ← D' signed w/ s' }

s, d don't leave Device 1

Bob can play sigchain from L1 to verify Alice





Defense against DDOS, corrupted Data

1) user signature chains are never rolled back ↗ prevents rollback attacks of using old keys
2) additions are signed and advertised → prevents fake "key updates" in that ∃ corroboration across multiple services & accounts

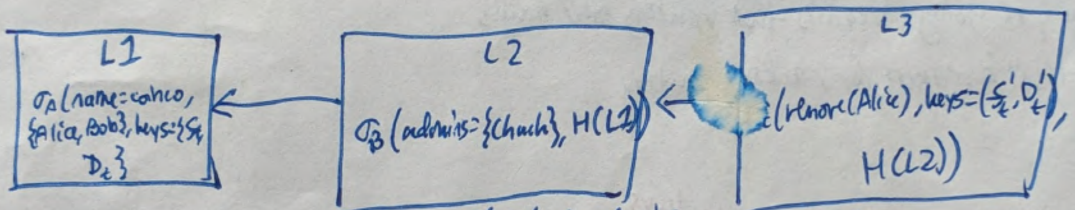
- prevents forking bc all sig chains must be captured in site's global Merkle Tree
  1) dl root, verify against site public key
  2) fetch signed root block, check chain against cached data
  3) post signature

Teams – team secret t

$(S_t, \mathcal{S}_t)$ (U_{Alice}, U_{BOB}) — Encrypts a t for U_A, U_B

(d_t, D_t)

sym-key for shared team data



Adding Chuck to admins

reroll keys once Alice is removed.

- Also when team members revoke devices, per-user keys reroll, and team must re-roll keys lec22 secure channels

A | B

$c = E(k, m)$
$h = MAC(k, m)$
$\xrightarrow{[c|h]}$
$m = D(k, c)$ — figure out m
$MAC(k, m) == h$? — Have to have k to get h

replay attacks? resending [c|h]

$c = E(k, m|seq)$
$h = MAC(k, m|seq)$
$\xrightarrow{[c|h]}$
$m|seq = d(k, c)$
$MAC(k, m|seq) == h$?

reflection attacks?

Eve $\xleftarrow{[c|h]}$

$C_a = E(k_a, m_a|seq_a)$
$h_a = MAC(k_a, m_a|seq_a)$
$\xrightarrow{[C_a|h_a]}$
$m_a|seq_a = D(k_a, c_a)$
$MAC(k_a, m_a|seq_a) == h_a$?

$c_b = E(k_b, m_b|seq_b)$
$h_b = MAC(k_b, m_b|seq_b)$
$\xleftarrow{c_b|h_b}$
$m_b|seq_b = D(k_b, c_b)$
$MAC(k_b, m_b|seq_b) == h_b$?

How do A, B know k_a, k_b, h?

random a $\longrightarrow g^a \bmod p \longrightarrow$ pick random b

$\xleftarrow{g^b \bmod p}$

$(g^b \bmod p)^a \bmod p$ | $(g^b \bmod p)^a \bmod p$

$k = g^{ab} \bmod p$

What if MITM?
- use key pairs

$c = E(k_a, m|seq_a)$
$h = MAC(k_a, m|seq_a)$
$Sig = sign(secret\_A, m|seq_a)$
$\xrightarrow{[c|h|sig]}$
$m|seq_a = D(k_a, c)$
$MAC(k_a, m|seq_a) == h$
$verify(m|seq_a, PK\text{-}a, sig) == yes$?

Lec 23 DDoS

* BOTNETS
* NIDS — anomaly based
  - signature based
  - Detection of Network Intrusion - blocks certain traffic

Attacks
1) HTTP Flooding
2) TCP SYN floods
3) optimistic ACKs
4) DNS amplification

- overloading routing tables
- hijacking prefixes

Lec 24 Blockchain

- distributed public logs

Questions
- Anonymity: users are public keys
- integrity: users sign their transactions
- preventing reordering - include hash of prev. txn

PKa:TX

H(PKa:TX)
PKB:TX

H(PKB:TX)
PKC:TX

Consensus
- take longest log
- make it hard to validate blocks

Lec 25 Tor

Anonymity!

Alice → P1 → P2 → P3 → S

from A to P1 [Data]

from P1 to P2 [Data]

from P2 to P3 [Data]

from P3 to S Data

circuit ID also encrypted + included

- Nothing says from: Alice; to: S
- Nothing receives a packet from Alice & directly sends it to S
- No one keeps state that links Alice to S
- data doesn't appear the same