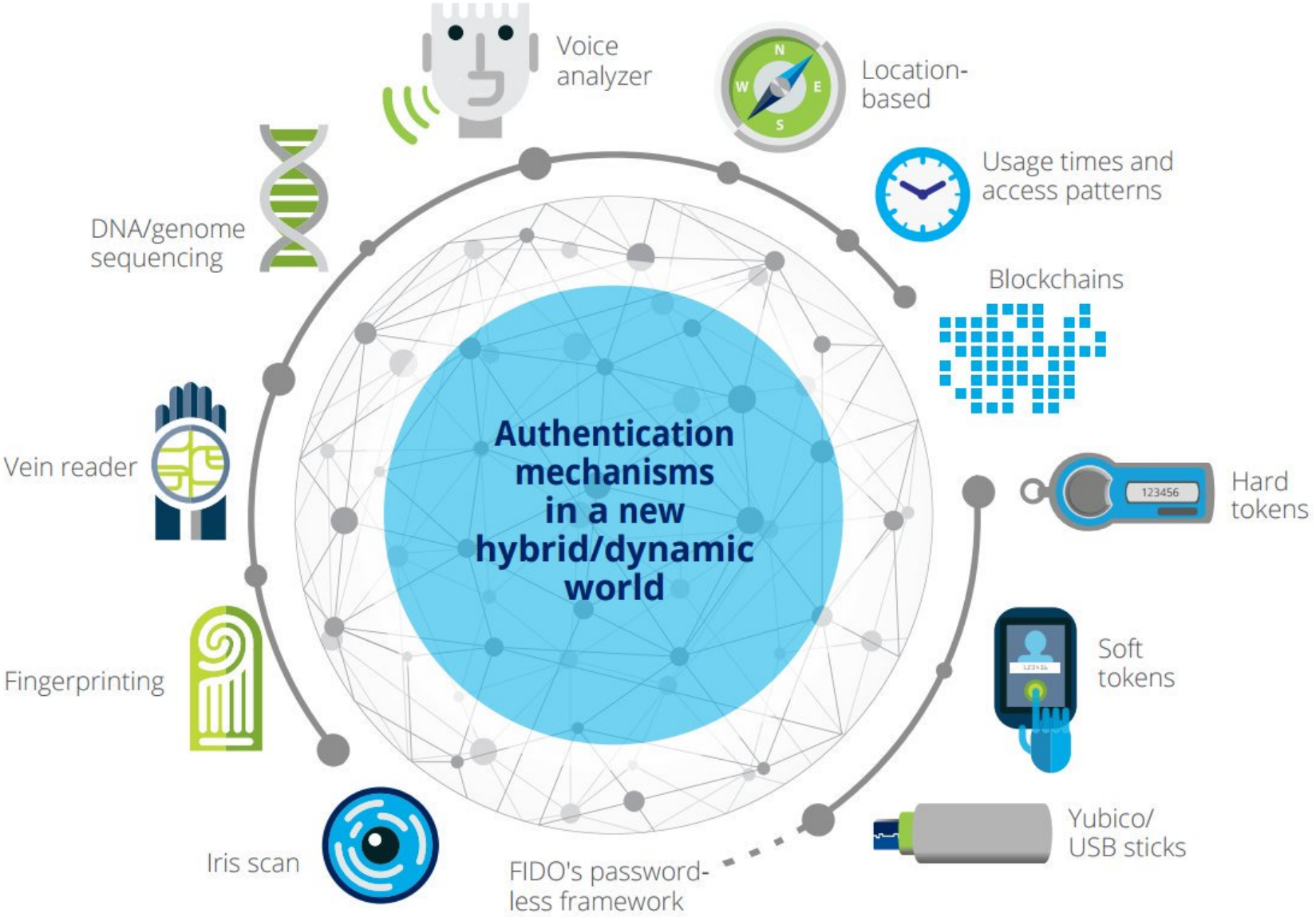
Beyond Passwords

Figure 3. A new world with many gatekeepers



Graphic: Deloitte University Press | DUPress.com

Beyond Passwords

- Users hate passwords 😡
- Security professionals hate passwords 😡
- Everybody hates passwords 😡
- Criteria to do better than passwords
  - Security
  - Usability
  - Deployability

Beyond Passwords

Security

- Physical observation
- Targeted impersonation
- Online or offline guessing
- Leaks
- Phishing
- Theft
- Trusted third party
- Privacy

Usability

- Memoryless
- Scalable for users
- Nothing to carry
- Physically effortless
- Easy to learn
- Efficient
- Infrequent errors
- Easy recovery from loss

Beyond Passwords

Deployability

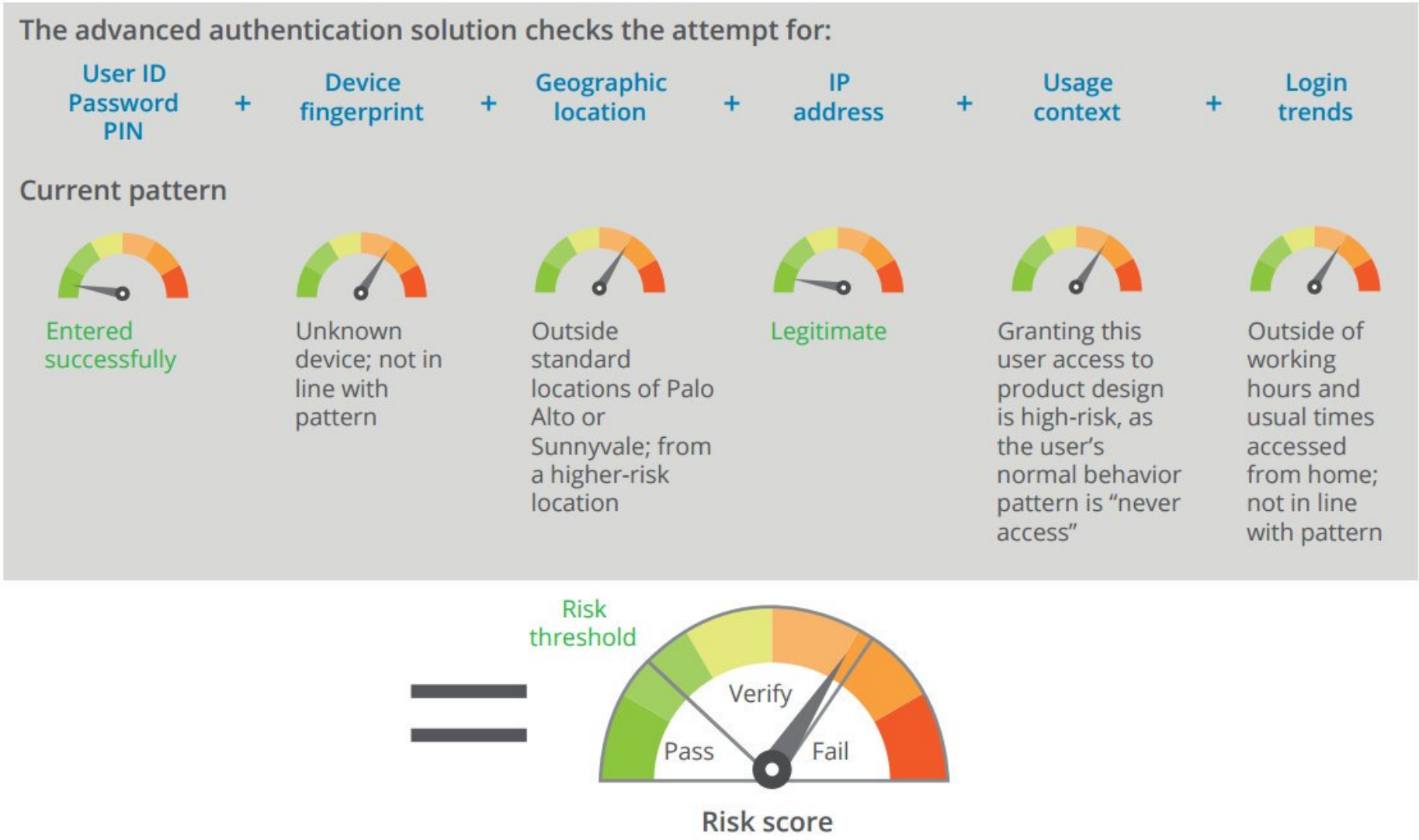
- Accessible
- Low Cost
- Server compatible
- Browser (client) compatible
- Mature
- Non-proprietary

Phew – long list!

Beyond Passwords

- Password managers
- Proxies
- Federated identity management
- Graphical
- Cognitive
- Paper tokens
- Visual cryptography
- Hardware tokens
- Phone-based
- Biometric

Figure 4. Risk-based user authentication



Graphic: Deloitte University Press | DUPress.com


PIZZA TIME THEATRE
CHUCK E. CHEESE
PIZZA TIME THEATRE
CHUCK E. CHEESE
SMILE AMERICA SAY
CHUCK E. CHEESE
PIZZA TIME THEATRE
CHUCK E. CHEESE
RALEIGH, N.C.
PIZZATIME THEATRE
CHUCK E. CHEESE'S

Tokens

Authentication Tokens

Authenticate a human based on possession of a small machine

Enrollment

- At enrollment, human is issued a token
  - Ranges from dumb (a physical key, a piece of paper) to a smart machine (a cryptographic processor)
  - Token becomes attribute of human's identity
- Easy to carry, maintenance-free, low cost
  - Only a subset of goals previously discussed!

Example: Garage Door Opener

- *Activated* by user (button press), provides entry past *barrier* (gate, door)
- One-pass protocol – only one message sent
- Token stores serial number *T*
- Barrier stores all serial numbers for all authorized tokens
- To enter: **Token->Barrier: T**

Example: Garage Door Opener

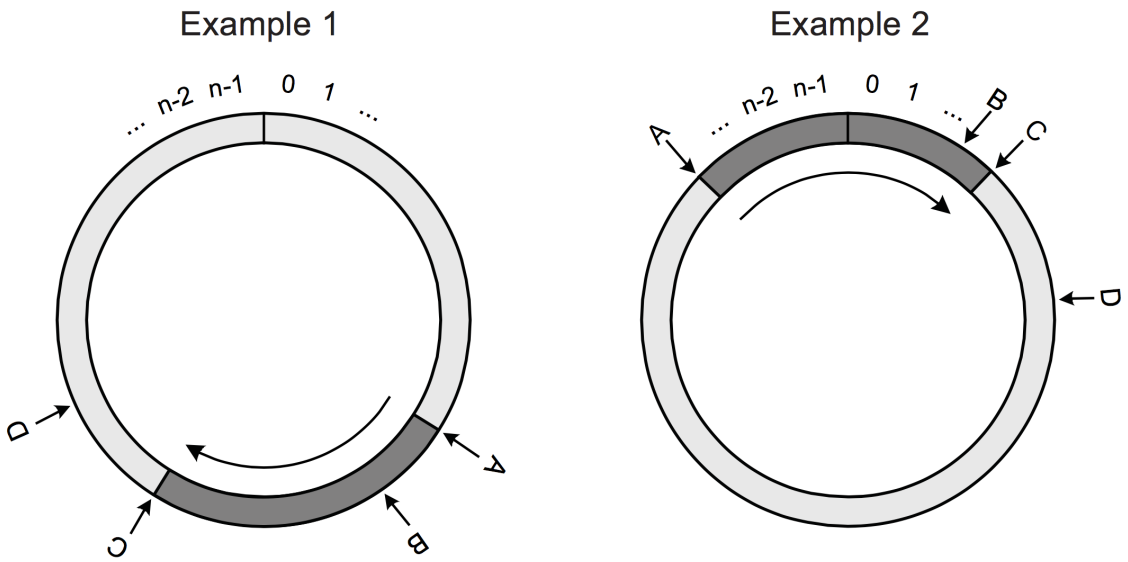
- Attack 1 – Replay attack
  - Thief waits nearby, captures serial number with antenna, programs new token with same number, gains entry
- Attack 2 – Brute force
  - Thief programs device to try all serial numbers (e.g. 16 bit numbers) and waits a little while to gain entry
- Countermeasure? **Nonce**

Example: Garage Door Opener

- Barrier has a (secret) master key – *mk*
- Token stores
  - Serial number *T*
  - Nonce *N* (sequence counter)
  - Shared key *k* which is H(*mk*, *T*)
- Barrier stores
  - Same values as token for all authorized tokens
  - Master key *mk*
- To enter: **Token→Barrier: T, MAC(T, N; k)**
  - Token increments *N*
  - Barrier increments *N* if MAC tag verifies

Example: Garage Door Opener

- Problem: Desynchronization of nonce
- Partial Solution: “Rolling window” of nonces



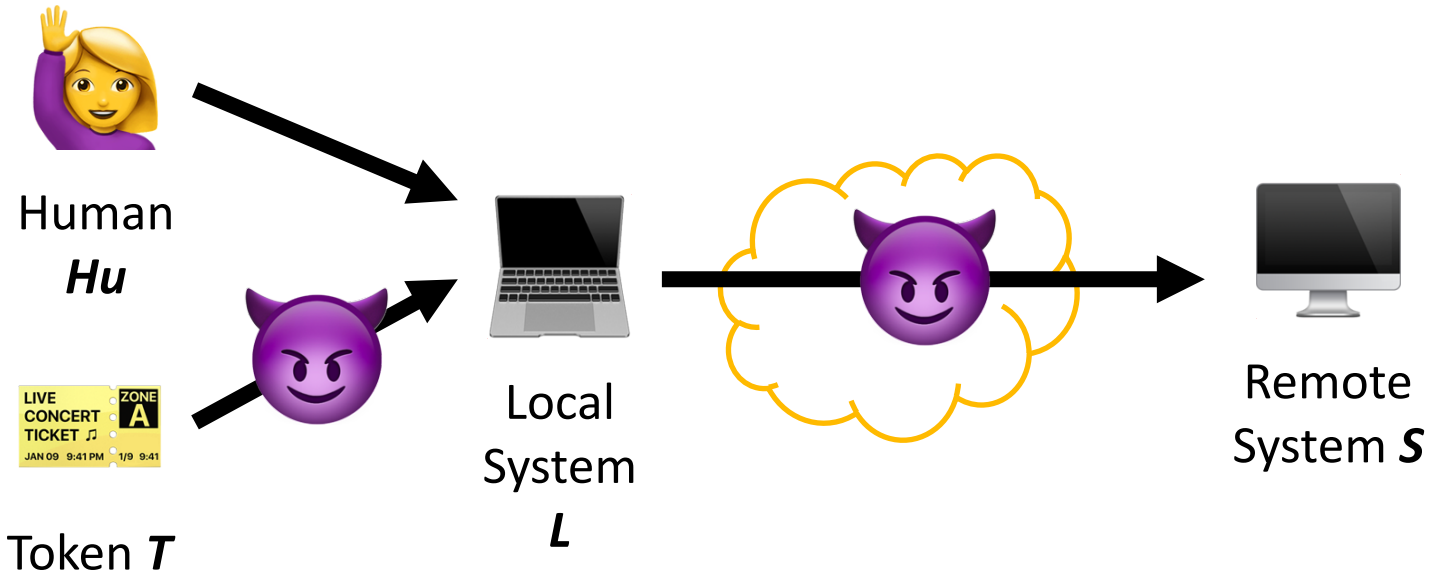
A - Value from last valid message

B - Accepted counter values

C - End of window

D - Rejected counter values

Remote Authentication



Example: SecurID

- Token displays *code* that changes every 60 seconds
  - LCD display
  - Internal clock
  - No human input
  - Can compute hashes and MACs
  - Stores secret (factory encoded random key)

- "nonce" is now current time (still a number used once)
- Uses local device (L) to input PIN

Example: SecurID

1. Hu→L: I want to authenticate as *id_Hu* to *S*
2. L and S: Establish secure channel (against Eve)
3. L→Hu: Enter pin and code on keyboard
4. T→Hu: code = MAC(time@T, id_T, kT)
5. Hu→L: pin, code
6. L: compute h=H(pin, code)
7. L→S: *id_Hu*, *h*
8. S: lookup (pin, *id_T*, *kT*) for *id_Hu*;
    1. id_Hu is authenticated if h=H(pin, MAC(*time@S*, *id_T*; *ktT*))

Assume

- Remote system *S* stores tuples (*id_T*, *id_Hu*, *kT*, pin)
- Local system *L*
- Human *Hu* stores PIN
- Token *T* stores *id_T*, *kT*

Example: SecurID

- Engineering challenge
  - Clock synchronization between *T* and *S*
  - *S* tracks clock skew on per-token basis
- Security challenge
  - Theft of *kT* from *S* for all tokens
    - 2011 data breach of RSA
    - Suspect that secure token seeds *may* have been stolen
    - RSA offered replacement tokens to 30,000 companies that used them

One-Time Password

```
...
~~50: MEND VOTE MALE HIRE BEAU LAY~~
~~49: PUG LYRA CANT JUDY BOAR AVON~~
48: LOAM OILY FISH CHAD BRIG NOV
47: RUE CLOG LEAK FRAU CURD SAM
46: COY LUG DORA NECK OILY HEAL
45: SUN GENE LOU HARD ELY HOG
44: GET CANE SOY NOR MATE DUEL
43: LUST TOUT NOV HAN BACH FADE
42: HOLM GIN MOLL JAY EARN BUFF
41: KEEN ABUT GALA ASIA DAM SINK
...
```

One-Time Password

- One-Time Password is only valid once (first use)
  - Similar to changing your password each time
  - Prevents replay attacks
  - Man-in-the-middle attacks still possible
- Use case: Login at untrusted public machine where keylogging is possible
- Use case: Account recovery if main password or two-factor device (e.g. phone) is lost
  - Google backup codes

One-Time Password

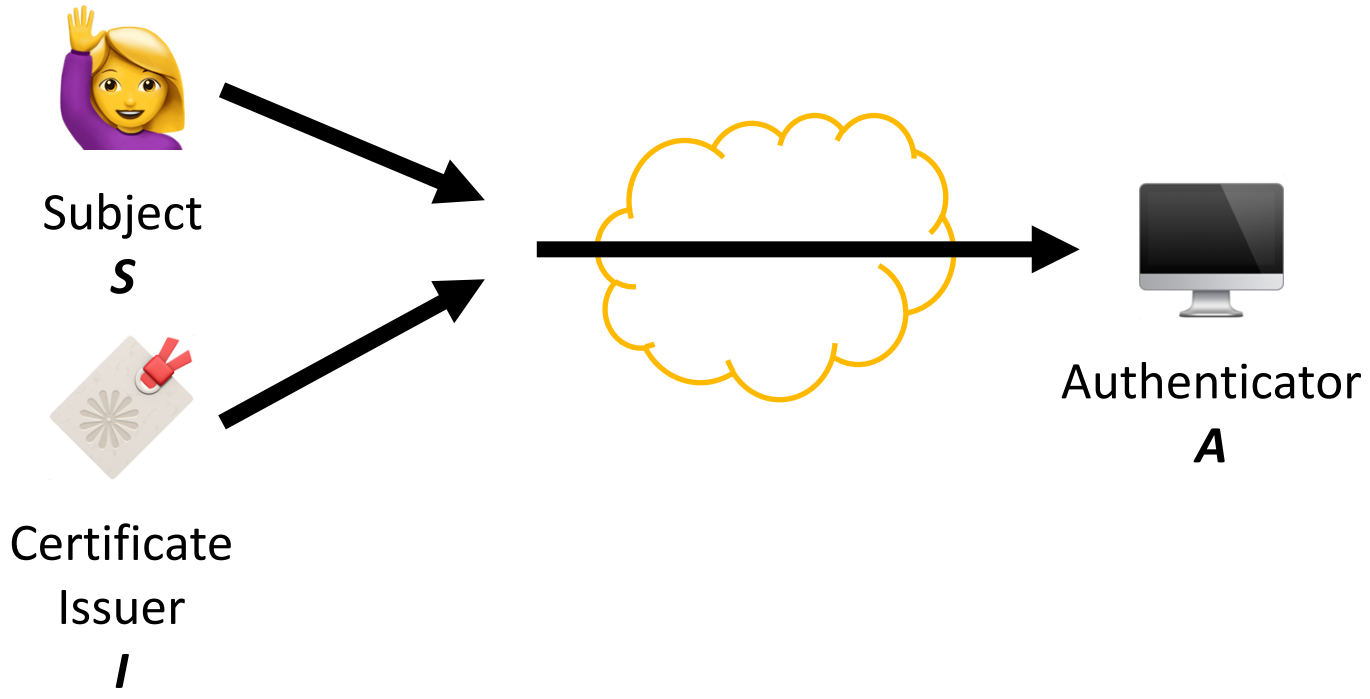
- Naïve-implementation
  - Pre-registered one-time passwords distributed on paper (hassle to obtain, risk of running out of passwords)
- Real implementation
  - Algorithm generation of one-time passwords
  - SecurID is an example – each code is a password valid for only 60 seconds
  - Generation method: *Iterated hashing*
    - Lamport's Scheme, S/KEY password system

Certificates

Digital Certificate

- **Digital certificate** binds together
  - Identity of principal
  - Public key of principal (encryption or verification key)
- Cert(S; I): Certificate issued by principal *I* for principal *S*
  - **Issuer** I certifies that K_S belongs to **Subject** id_S
  - b = id_S, K_S *(id of subject, key of subject)*
  - Cert(S; I) = b, Sign(b; k_I)
- Fingerprint: H(Cert(S; I))

Digital Certificate Authentication



Digital Certificate Authentication

1. S: Let msg = "I'm id_S".
   Compute s = Sign(m; k_S)
2. S →A: msg, s
3. A: Find Cert(S; I)
   1. Verify I's signature on cert
   2. Verify id_S
   3. Retrieve K_S
   4. Accept if Ver(msg; s; K_S).

- Notes
  - I must be trusted to issue certificate
  - A must verify id_S – don't omit!

X.509 Certificates

Overview

- Standard format for certificates
  - RFC 5280
- Used for
  - SSL/TLS
  - S/MIME (email)
  - EAP-TLS (Wi-Fi)

Contents

- Serial number
- Issuer *distinguished name*
- Validity period (start and end time)
- Subject *distinguished name*
- Subject public key (and name of algorithm)
- Issuer's signature for all above data (and name of algorithm)

X.500 Distinguished Names

- General purpose directory
- Common options for X.509 certificates
  - Common Name (CN): Human full name, server name, or domain name
  - Organization unit (OU): Finance, HR, …
  - Organization (O): Pacific, Google, …

Certificate Chain

- Problem
  - Receive a message signed by A, but don't know A's public verification key
  - Find a certificate Cert(A; B)
    - Certificate for A signed by B
  - But don't know B's public key either!

Certificate Chain

- Solution: **Recursion** ☺
  - Set of certificates
  - Cert(A; B), then Cert(B; C); then Cert(C; D), and hopefully you know D's public key
- **Certificate chain** – sequence of certificates that certify each other
  - One end: Certificate for principal you want to authenticate
  - Other end: Certificate for principal you already know
    - Root or anchor of trust
  - Must trust every issuer in the chain to issue certificates

Public-Key Infrastructure

- System for managing distribution of certificates
- Two models
  - Decentralized – peer to peer, no leader
    - PGP
  - Centralized – oligarchy, leadership by elite
    - CA

PKI Decentralized: PGP

PKI Example: PGP

- "Pretty Good Privacy"
  - Encryption tool for emails and files
  - Dates back to early days of crypto (1991)
  - Developed by Phil Zimmermann
    - Investigated by US Government for "Munitions export without a license"

PKI Example: PGP

- Each user manages a *keyring*
- Alice has her key in her keyring
- Alice meets Bob at *key-signing party*
  - She copies his key into her keyring
  - She marks Bob as *fully* or *marginally trusted* as an *introducer*
  - She copies other keys he might have collected, too
- Other option: Downloading keys from a key server (but you have little proof of who they actually belong to)

PKI Example: PGP

Never bring tequila to a key-signing party….

https://xkcd.com/364/

PKI Example: PGP

- Entries on the keyring are certificates
- Alice's own key on her keyring
  - Cert(A; A) <- Self-signed certificate!
- When Alice imports a key signed by Bob, she gets Cert(C; B)
  - She can import as-is and put Cert(C; B) into keyring
  - She can vouch for it and put Cert(C; A) into keyring
- Can phone Bob and manually verify a certificate taken from a key server

PKI Example: PGP

- Keys on keyring are *fully valid* only if
  - Signed by 1 fully trusted introducer or 3 marginally trusted introducers
  - The certificate chain leading from key to user's own key has length of 5 or less
- Valid keys can be used for encryption and signing

PKI Example: PGP

"As time goes on, you will accumulate keys from other people that you may want to designate as trusted introducers. Everyone else will each choose their own trusted introducers. And everyone will gradually accumulate and distribute with their key a collection of certifying signatures from other people, with the expectation that anyone receiving it will trust at least one or two of the signatures. This will cause the emergence of a **decentralized fault-tolerant web of confidence** for all public keys."

Phil Zimmermann, 1992

PKI Centralized: CAs

PKI Example: CAs

- Certificate Authority (CA)
  - Principal whose purpose is to issue certificates
- Centralized PKI philosophy

PKI Example: CAs

- Everyone enrolls with CA to get certificate
  - Example: Alice enrolls and get Cert(Alice; CA)
- Bob's system comes pre-installed with CA's self-signed certificate Cert(CA; CA)
- When Bob receives message signed by Alice
  - Bob contacts CA to get Cert(Alice; CA)
  - Or Alice includes that certificate with her message

PKI Example: CAs

- Web server has Cert(server; CA) installed
  - Server identity is hostname
  - CA is a root for which Cert(CA; CA) is installed in browser
- Browser authenticates web server using hostname and public key from certificate

Many Certificate Authorities

- Many **many** Certificate Authorities
  - No single CA will be trusted by all world governments, militaries, businesses, …
- OS and web browsers come with some CAs pre-installed
- Organizations act as their own CAs
  - Company issues certificates to employees for VPN
  - Central bank issues certificates to other banks
  - Manufacturer issues certificates to sensing devices

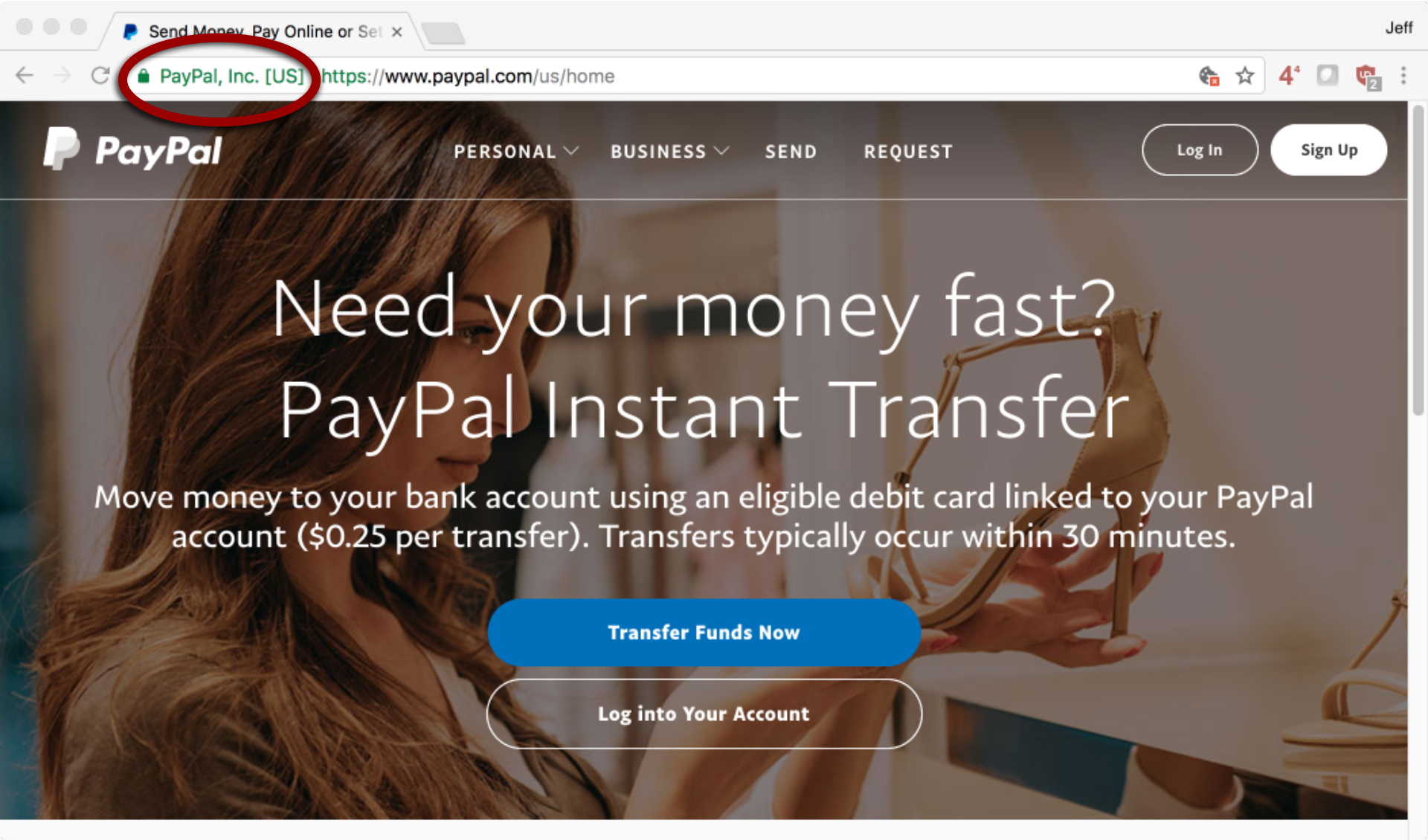
Enrollment with CA

- You create a key pair – the CA never knows your private key
- You generate a *certificate signing request (CSR)* containing the identity you are claiming
- You send the CSR to a CS (w/payment?)
- CA verifies your identity (how well?)
- CA signs your public key, creating a certificate, and sends certificate to you

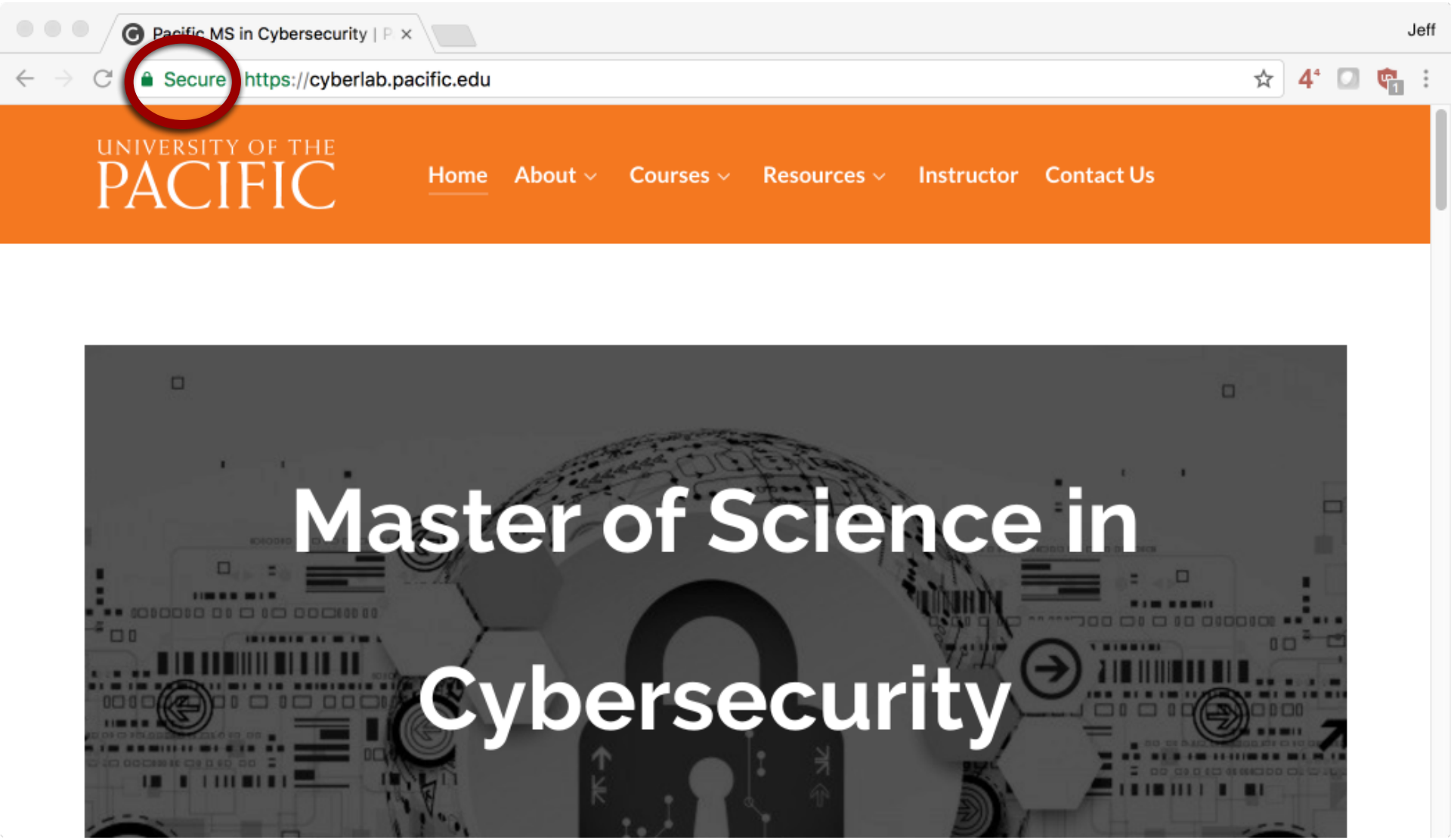
Identity Verification

- *Extended Validation* (EV) certificate
  - CA does extra checking of your identity
  - Certificate marked as having received EV
  - Web browser displays EV mark in GUI
- Extra checking *(in exchange for more $$$)*
  - Verify legal existence of organization
  - Verify physical presence of organization
  - Verify ownership/control over domain
- CA records that data in certificate as part of subject identity

Extended Validation Certificate



Domain Validated Certificate



Issuing Certificates

- Conflicting goals
- CA private signing key must be kept *secret*
  - Public verification key is pre-installed on user systems and hard to update
  - A leaked private signing key could forge certificates
  - Solution: Keep private key offline in “cold storage”
- CA private signing key must be *available* for use
  - Needed to sign new certificates for customers
  - Solution: Keep it in computer memory

Issuing Certificates

- Solution? Use root and intermediate CAs
- **Root CA**
  - Certificate at root of trust in chain
  - Public key pre-installed at client PCs
  - Private key kept offline / highly secure
- **Intermediate CAs**
  - Certified by root CA
  - Used to certify user keys
  - Might by run by different organization than root CA

PKI Problems

Problem 1: Revocation

- Key gets compromised (subject or issuer)
  - Your website gets hacked and private key stolen
- Subject leaves an organization (and certificates need to be revoked)
- Several (mediocre) options
  - Fast expiration
  - Certificate revocation list (CRL)
  - Online certificate validation

Problem 1: Revocation

- **Fast expiration**
- Idea
  - Validity interval is short (10 mins to 24 hours)
  - Any compromise is for a bounded time period
- Problems
  - CAs have to issue new certificates frequently (do they need to re-check identity?)
  - Machines have to update certificates frequently
    - Would need to automate

Problem 1: Revocation

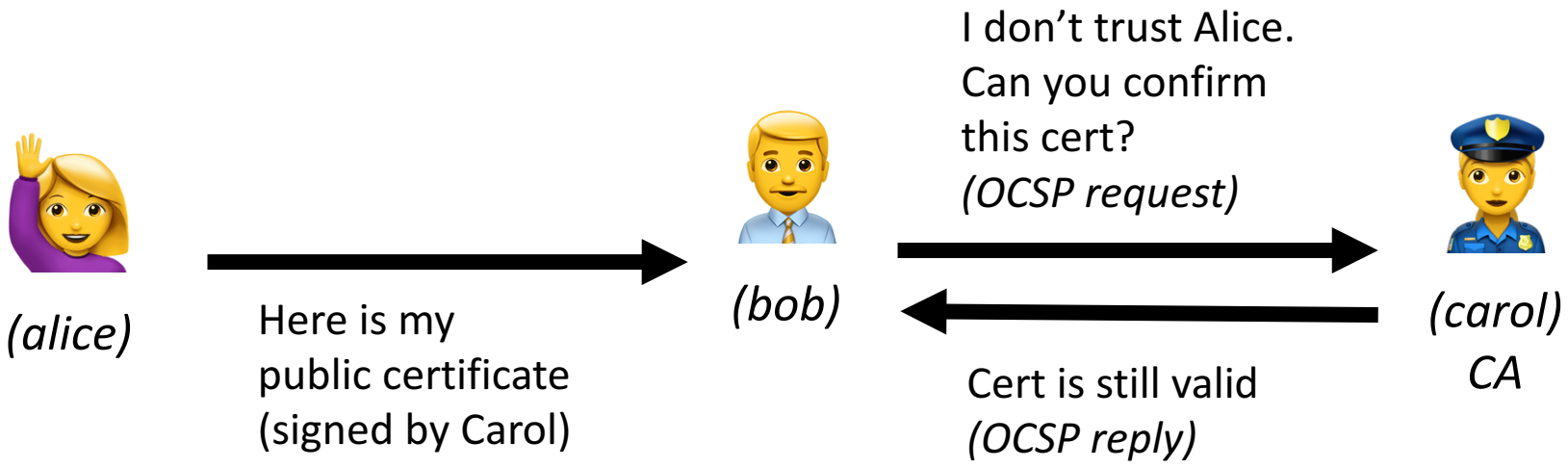
- **Certificate Revocation List (CRL)**
- Idea
  - CA posts lists of revoked certificates
  - Clients download and check list every time they need to validate certificate
- Problems
  - Clients don't bother checking (usability problems)
    - Large list, download time
  - Or clients cache (TOCTOU attack)
  - CRL must always be available (DDOS attack?)
- Chromium example – limit of 250kB
  - https://dev.chromium.org/Home/chromium-security/crlsets

Problem 1: Revocation

- **Online Certificate Validation**
- Idea
  - CA runs validation server
  - Client contacts server each time to validate certificate
- Problems
  - Clients don't bother
    - Checking adds latency to each new session
  - Server must always be available (DDOS?)
    - Clients "soft fail" to mitigate risk and users don't notice
  - Reveals to CA which websites you want to access - privacy

Problem 1: Revocation

- Online Certificate Status Protocol (OCSP)
  - Support: IE, Firefox, Safari, but not Chrome



Note: Alice and Bob both trust Carole as CA. (Have Carole's root cert preinstalled)

Problem 1: Revocation

- **OCSP Stapling (aka TLS Certificate Status Request)**
- Idea
  - Certificate must be accompanied by “fresh” assentation from CA that certificate is valid (window of a few days)
  - Whoever presents certificate to client is *also* responsible for acquiring the fresh assertation and *stapling* it to the certificate
- Bypasses most problems with online validation
  - No privacy concern – the CA only knows their customer *(website)*, not the client *(visitor)*
  - Performance better – Certificate holder requests verification once (per time interval) – no need for each client to verify!
  - Clients don’t incur latency of verification request
- Support: Firefox, IE, Chrome

Problem 1: Revocation

- https://blog.hboeck.de/archives/886-The-Problem-with-OCSP-Stapling-and-Must-Staple-and-why-Certificate-Revocation-is-still-broken.html
  - Still not perfect! (as of early 2017)
  - Implementation issues with Apache and Nginx make it risky to enable OCSP Stapling without risking your clients receiving errors in the case of temporary failure of OCSP verification server
- Doesn't work with intermediate CA certs (can only staple one OCSP response at a time)

Problem 2: Authority

- CA goes rogue or gets hacked
  - *Already discussed in cryptography discussion*
- Mediocre solutions
  - **HTTP Public Key Pinning (HPKP)**
    - Upon first connection to server, client leans of public keys. In future connections, certificate must contain one of those keys
    - Deprecated by Chrome – too risky to deploy!
  - **Certificate transparency**
    - Maintain public log of issued certificates and monitor log to detect malicious activity
  - **DNS Certificate Authority Authorization (CAA)**
    - DNS record for entity specifies list of allowed CAs
    - For the CA, not for the client! Legitimate CA won't issue cert unless in DNS list
  - **DNS-based Authentication of Named Entities (DANE)**
    - Bypasses CAs entirely and relies on DNS to bind certificates to host names