Authentication Mechanisms and Protocols

Authentication (Authn)

Proof that an entity has an attribute it claims to have

- Hi, I’m Joe
- Prove it!
- Here are my Joe’s credentials
- Credentials accepted/not accepted

- Hi, I’m over 18
- Prove it!
- Here is the proof
- Proof accepted/not accepted

Authn: Proof Types

Something we know

- A secret memorized (or written down…) by Joe

Something we have

- An object/token solely held by Joe

Something we are

- Joe’s Biometry

Multi-factor authentication

- Simultaneous use of different proof types
- 2FA = Two Factor Authentication

Authn : Goals

Authenticate interactors

- People, services, servers, hosts, networks, etc.

Enable the enforcement of authorization policies and mechanisms

- Authorization != authentication
- Authorization $\Rightarrow$ authentication

Facilitate the exploitation of other security-related protocols

- e.g. key distribution for secure communication

Authn : Requirements

- **Trustworthiness**
  - How good is it in proving the identity of an entity?
  - How difficult is it to be deceived?
  - Level of Assurance (LoA)

- **Secrecy**
  - No disclosure of secret credentials used by legitimate entities

NIST 800-63

LoA	DESCRIPTION	TECHNICAL REQUIREMENTS		
		IDENTITY PROOFING REQUIREMENTS	TOKEN (SECRET) REQUIREMENTS	AUTHENTICATION PROTECTION MECHANISMS REQUIREMENTS
1	Little or no confidence exists in the asserted identity; usually self-asserted; essentially a persistent identifier	Requires no identity proofing	Allows any type of token including a simple PIN	Little effort to protect session from off line attacks or eavesdropper is required.
2	Confidence exists that the asserted identity is accurate; used frequently for self service applications	Requires some identity proofing	Allows single-factor authentication. Passwords are the norm at this level.	On-line guessing, replay and eavesdropping attacks are prevented using FIPS 140-2 approved cryptographic techniques.
3	High confidence in the asserted identity's accuracy; used to access restricted data	Requires stringent identity proofing	Multi-factor authentication, typically a password or biometric factor used in combination with a 1) software token, 2) hardware token, or 3) one-time password device token	On-line guessing, replay, eavesdropper, impersonation and man-in-the-middle attack are prevented. Cryptography must be validated at FIPS 140-2 Level 1 overall with Level 2 validation for physical security.
4	Very high confidence in the asserted identity's accuracy; used to access highly restricted data.	Requires in-person registration	Multi-factor authentication with a hardware crypto token.	On-line guessing, replay, eavesdropper, impersonation, man-in-the-middle, and session hijacking attacks are prevented. Cryptography in the hardware token must be validated at FIPS 140-2 level 2 overall, with level 3 validation for physical security.

Authn : Requirements

Robustness

- Prevent attacks to the protocol data exchanges
- Prevent on-line DoS attack scenarios
- Prevent off-line dictionary attacks

Simplicity

- It should be as simple as possible to prevent entities from choosing dangerous shortcuts

Deal with vulnerabilities introduced by people

- They have a natural tendency to facilitate or to take shortcuts

Authn:
Entities and deployment model

Entities

People

Hosts

Networks

Services / servers

Deployment model

Along the time

- Only when interaction starts
- Continuously along the interaction

Directionality

- Unidirectional
- Bidirectional (Mutual)

Authentication interactions: Basic approaches

Direct approach

1. Provide credentials
2. Wait for verdict

Challenge-response approach

1. Get challenge
2. Provide a response computed from the challenge and the credentials
3. Wait for verdict

Authentication of subjects: Direct approach w/ known password

A password is checked against a value previously stored

- For a claimed identity (username)

Personal stored value:

- Transformed by a unidirectional function
- Windows: digest function
- UNIX: DES hash + salt
- Linux: MD5 + salt
  - hash is configurable

Optimal: PBKDF2, Script with high complexity

Authentication of subjects: Direct approach w/ known password

Advantage

- Simplicity!

Problems

- Usage of weak keys
  - They enable dictionary attacks
- Transmission of passwords along insecure communication channels
  - Eavesdroppers can easily learn the password
  - e.g. Unix remote services, PAP

Top Ten 2017 from Splashdata

1. 123456
2. Password
3. 12345678
4. qwerty
5. 12345
6. 123456789
7. letmein
8. 1234567
9. football
10. iloveyou

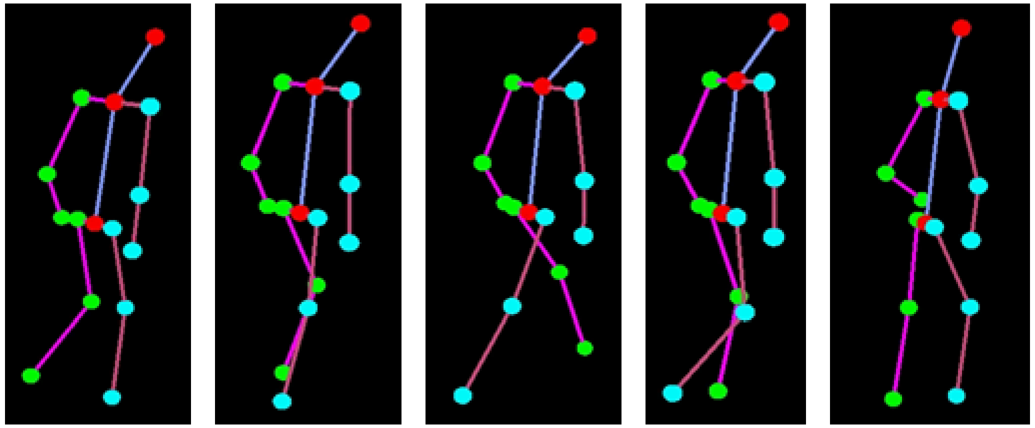
Authentication of subjects: Direct approach with biometric

People get authenticated using body measures

- Biometric samples
- Fingerprint, iris, face geometry, voice timber, manual writing, vein matching, etc.

Measures are compared with personal records

- Biometric references (or template)
- Registered in the system with a previous enrolment procedure



Authentication of subjects: Direct approach with biometrics

Advantages

- People do not need to use memory, or carry something
  - Just be their self
- People **cannot** choose weak passwords
  - In fact, they don’t chose anything
- Authentication credentials cannot be transferred to others
  - <u>One cannot delegate its own authentication</u>

Authentication of subjects: Direct approach with biometrics

Problems

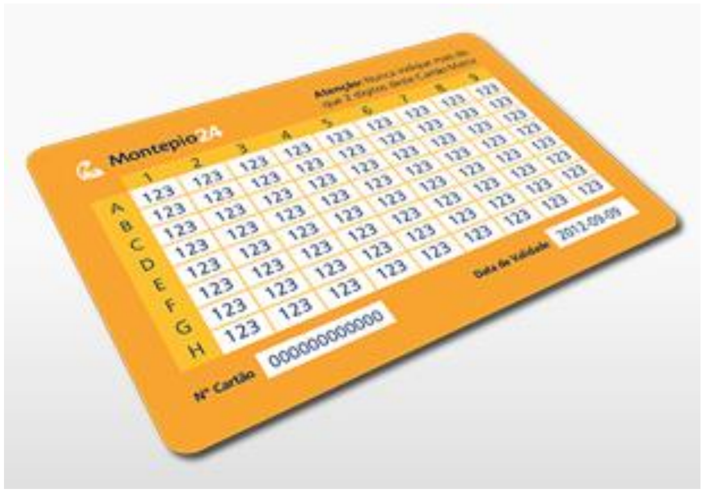
- Biometric methods are still incipient
  - In many cases it can be fooled with ease (Face Recognition, Fingerprint)
- People cannot change credentials
  - If the credentials or templates are stolen
- Credentials cannot be transferred between individuals
  - If it is required in extraordinary scenarios
- Can pose risks to individuals
  - Physical integrity can be compromised by an attacker in order to acquire biometric data
- It is not easy to be implemented in remote systems
  - It is mandatory to have secure and trusted biometric acquisition devices
- Biometrics can reveal other personal secrets
  - Diseases

Authentication of subjects: Direct approach with one-time passwords

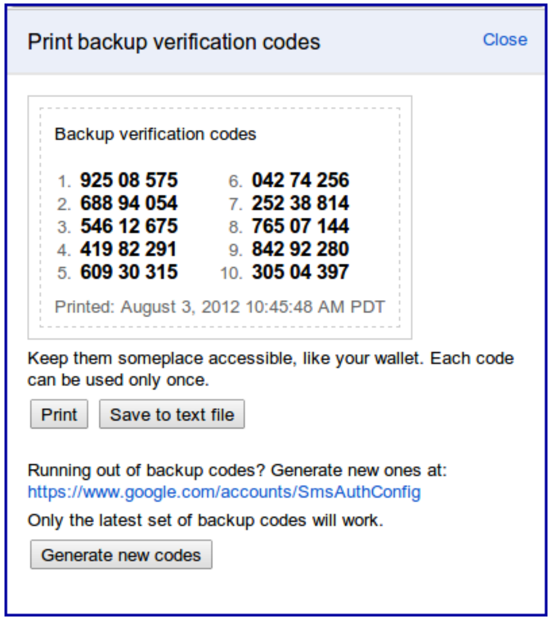
One Time Passwords = Secrets that can be used only once

- Pre-distributed directly, or the result of a generator function

Example: Bank codes, Google Backup Codes



https://www.montepio.pt/SitePublico/pt_PT/particulares/montepio24/cartao-matriz.page?altcode=10006P



Print backup verification codes

Close

Backup verification codes

1. 925 08 575	6. 042 74 256
2. 688 94 054	7. 252 38 814
3. 546 12 675	8. 765 07 144
4. 419 82 291	9. 842 92 280
5. 609 30 315	10. 305 04 397

Printed: August 3, 2012 10:45:48 AM PDT

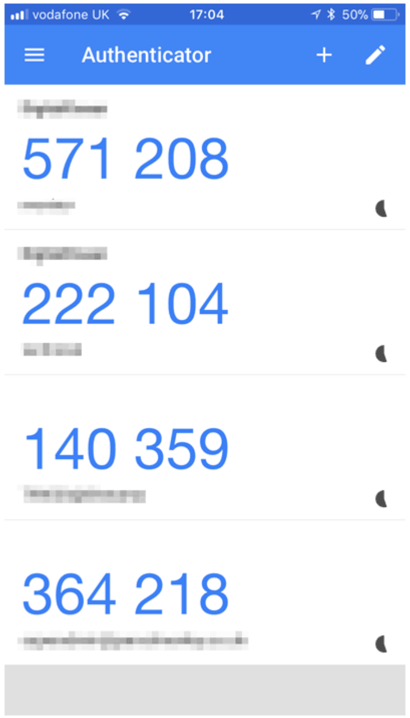
Keep them someplace accessible, like your wallet. Each code can be used only once.

Print · Save to text file

Running out of backup codes? Generate new ones at: https://www.google.com/accounts/SmsAuthConfig

Only the latest set of backup codes will work.

Generate new codes



Authentication of subjects: Direct approach with one-time passwords

Advantages

- Can be eavesdropped, allowing its use in channels without encryption
- Can be chosen by the authenticator, which may enforce a given complexity
- Can depend on a shared password

Problems

- Interacting entities need to know which password to use in each occasion
  - Implies some form of synchronization (e.g, index, coordinates)
- Individuals may require additional resources to store or generate the passwords
  - Sheet of paper, application, additional device, etc.

RSA SecurID

Personal Authentication Device

- Can also exists as software modules for mobile devices (smartphones)

Generate a unique number at fixed intervals

- Usually one per minute or per 30 seconds
- Sequence is associated to a individual (User ID)
- Number is calculated by considering:
  - A 64 bit secret key stored in the device
  - The current date and time
  - A proprietary algorithm (SecurID hash)
  - Optionally: a PIN code

RSA SecurID

Authentication with Unique Keys

Subjects generates an OTP by combining his User ID with the number presented by the device

- OTP = User ID, Token Number

The RSA ACE Server does the same: given an User ID it calculates the Token, and check if they match

- Server knows the User ID and the 64 bits shared key
- Server and token have their clocks synchronized. Additional measures must be taken to deal with the clock skew.
  - RSA Security Time Synchronization

Robust against dictionary attacks

- Keys are not chosen by individuals

Vulnerable to attacks to the RSA ACE Server

- 2011: Adobe Flash Zero Day exploited from Flash object in XLS spreadsheet

Yubikey

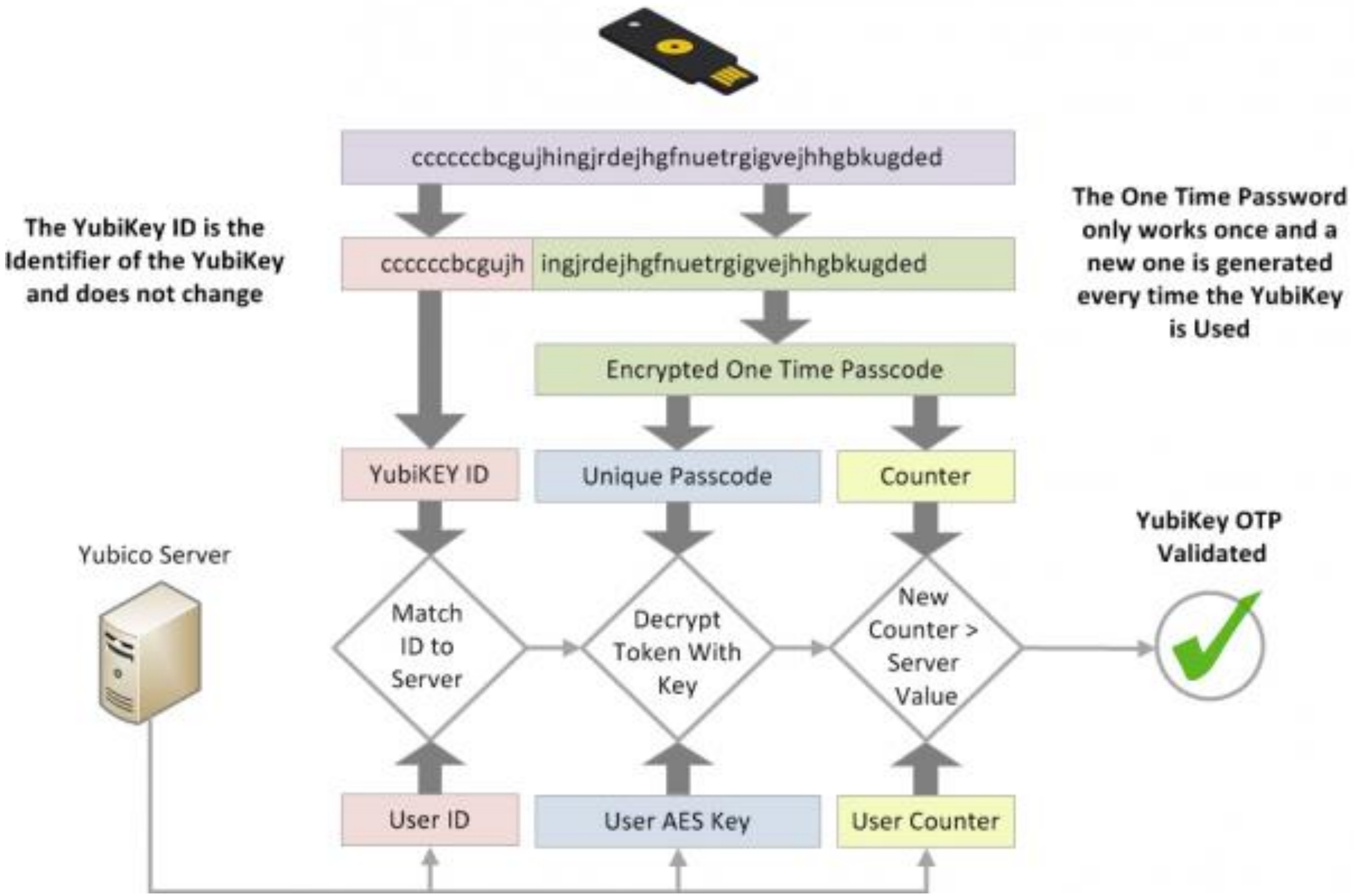
Personal Authentication Device

- USB and/or NFC

Activation generates a 44 characters key

- Emulates a USB keyboard (besides an own API)
- Supports HOTP (events) or TOPT (Temporal)
- If a challenges is provided, user most touch the button to obtain a result
- Several algos, including AES 256

cccjgjgkhcbbirdrfdnlnghhfgrtnnlgedjlftrbdeut


cccccbcgujhingjrdejhgfnuetrgigvejhhgbkugded
cccccbcgujh
ingjrdejhgfnuetrgigvejhhgbkugded
The YubiKey ID is the Identifier of the YubiKey and does not change
The One Time Password only works once and a new one is generated every time the YubiKey is Used
Encrypted One Time Passcode
YubiKEY ID
Unique Passcode
Counter
Yubico Server
YubiKey OTP Validated
Match ID to Server
Decrypt Token With Key
New Counter > Server Value
User ID
User AES Key
User Counter

Challenge Response Approach

The authenticator provides a challenge (e.g, a NONCE)

The authenticated entity transforms the challenge
- The transformation method is shared with the authenticator

The result is sent to the authenticator

The authenticator verifies the result
- Calculates a result using the same method and challenge
- or... produces a value from the result and evaluates if it is equal to the challenge, or to some related value

Challenge Response Approach

Advantages

- Authentication credentials are not exposed
- An eavesdropper will see the challenge and the result, but has no knowledge about the transformation

Problems

- Authenticated entities must have the capability of calculating results to challenges
    - Hardware token ou software application
- The authenticator may need to keep shared secrets (in clear text)
    - Secrets can be stolen
    - Individuals may reuse secrets in other systems, enabling lateral attacks.
- May be possible to calculate all results to a single (or all) challenge(s)
    - Can revel the secret used
- May be vulnerable to dictionary attacks
- Authenticator should **NEVER** issue the same challenge to the same user.

Authentication of Subjects: Challenge response with Smartcards

Authentication Credentials

- Having the SmartCard
  - e.g., the Citizen Card
- The private key stored inside the smartcard
- The PIN code to access the key

The authenticator knows

- The user public key

Robust against:

- Dictionary attacks
- Offline attacks to the database
- Insecure channels

PORTUGAL
SPECIMEN
I<PRT00000000D<<0<ZZ4<<<<<<<<<<2
8108108F1201158PRT<<<<<<<<<<<1
AVILA<<PAULA<ANDREIA<CONCEICAO

Authentication of Subjects: Challenge response with Smartcards

Challenge Response Protocol

- The authenticator generates a random challenge
  - Or a value that was never used before (NONCE)
- SmartCard owner ciphers the challenge with his private key
  - Stored in the smart card, protected by the PIN code
  - In alternative, he can sign the challenge
- The authenticator deciphers the result with the private key
  - If the decrypted result matches the challenge, the authentication is successful.
  - In alternative, it can verify the signature (which is the same process)

Authentication of Subjects: Challenge response with Shared Secret

Authentication Credentials

- Password selected by the individual

The authenticator knows:

- Bad approach: the shared password
- Better approach: A transformation of the shared password
  - The transformation should be unidirectional

Authentication of Subjects: Challenge response with Shared Secret

Basic Challenge-Response Protocol

- The authenticator generates a random value
  - Or a value that was never used before (NONCE)
- The individual calculates a transformation of the challenge and the password
  - result = hash(challenge || password)
  - or... result = encrypt(challenge, password)
- The authenticator reverts the process and check if the values match
  - result == hash(challenge || password)
  - or .... challenge == decrypt(result, password)
- Examples: CHAP, MS-CHAP v1/v2, S/Key

PAP and CHAP (RFC 1334, 1992, RFC 1994, 1996)

Protocols user for PPP (Point-to-Point Protocol)

- Unidirectional authentication
  - The authenticator authenticates users, but users do not authenticate the authenticator

PAP (PPP Authentication Protocol)

- Simple presentation of a UID/Password pair
- Insecure transmission (in clear text)

CHAP (CHallenge-response Authentication Protocol)

Aut $\rightarrow$ U : authID, challenge

U $\rightarrow$ Aut: authID, MD5(authID, secret, challenge), identity

Aut $\rightarrow$ U : authID, OK/not OK

- The authenticator can request further authentication at any time

S/Key (RFC 2289, 1998)

Authentication Credentials: A password

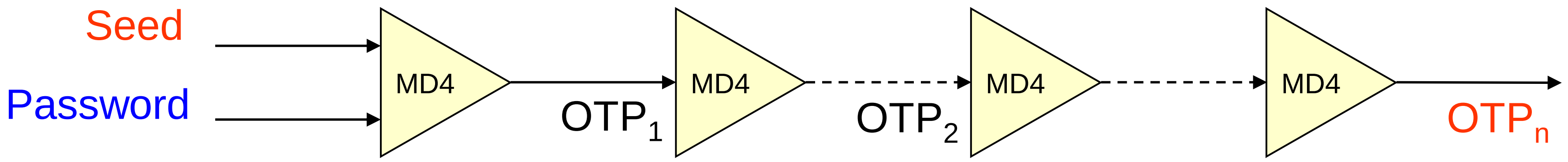
The authenticator knows

- The last One-Time Password (OTP) that was used
- The index of the last OTP used
  - Defines an order between consecutive OTPs
- A seed (or root) of all OTPs

Authenticator setup process

- The authenticator defines a random seed
- The individual generates the initial OTP:
  - **$OTP_n = h_n$ (seed, password), where h = MD4**
  - Alternative versions of S/Key use MD5, SHA-1 or other
- The authenticator stores the seed, the index N and OTPn, to use in further authentication processes

S/Key (RFC 2289, 1998)



S/Key: Authentication Process

The authenticator sends the seed and the index for that specific user

- They are considered a challenge

The user will generate index-1 consecutive OTPs

- Uses the last one ($OTP_{index-1}$) as the result to the challenge presented

The authenticator calculates h(result) and compares the value with OTP_{index} that is stored

- If h(result) = OTP_{index}, the authentication is successful
- If the process is successful, it stores the last values used for the index and the OTP
  - index-1 e $OTP_{index-1}$

Authentication of subjects: Challenge-Response with shared secret

Uses a cryptographic shared key instead of a password

- Robust against dictionary attacks
- Requires a device to store the shared key

GSM: Authentication of a Subscriber

Based on a secret shared between the HLR and the subscriber phone

- Uses 128 bit shared key (not a asymmetric key pair)
- Key is stored in the SIM card
- SIM card answers challenges using the shared key

Uses (initially unknown algorithms):

- A3 for authentication
- A8 to generate the session key
- A5 is a stream cipher for communication

A3 and A8 are executed by the SIM, A5 executed by the baseband

- A3 and A8 can be chosen by the operator

GSM: Authentication of a Subscriber

MSC requests triples from HLR/AUC

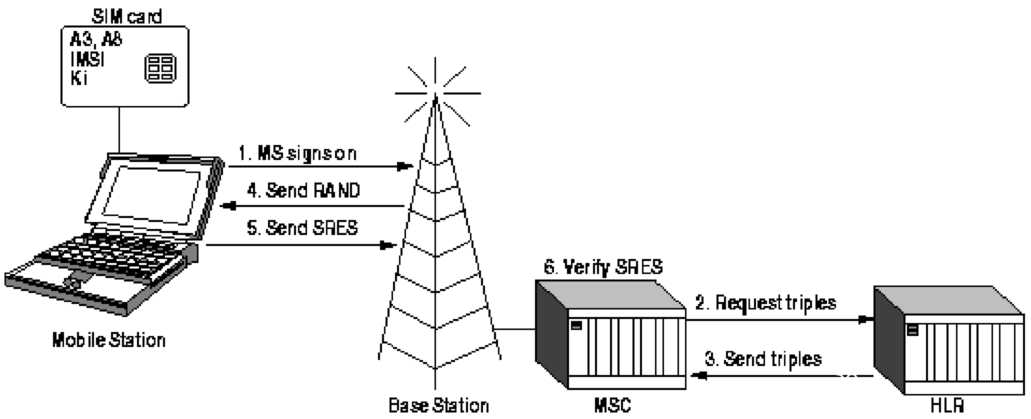
- RAND, SRES, Kc
- he can ask one or several

HLR generates RAND and the triples using the subscriber Ki

- RAND, random value (128 bits)
- SRES = A3 (Ki, RAND) (32 bits)
- Kc = A8 (Ki, RAND) (64 bits)

Frequently uses COMP128 for the A3/A8 algorithms

- Recommended by the GSM consortium
- [SRES, Kc] = COMP128 (Ki, RAND)



Authentication of Systems

By name (DNS) or MAC/IP address

- Extremely weak, without cryptographic proof
  - Still... it is used by some services
  - e.g., NFS, TCP wrappers

With cryptographic keys

- Secret keys, shared between entities that communicate frequently
- Asymmetric key pairs, one per host
  - Public keys pre-shared with entities that communicate frequently
  - Public keys certified by a third party (a CA)

Authentication of Services

Authentication of the host

- All services co-located in the same host are automatically and indirectly authenticated

Credentials exclusive to each service

Authentication:

- Secret keys shared with clients
  - When they require authentication of the clients
- Asymmetric key pairs by host/service
  - Certified by others or not

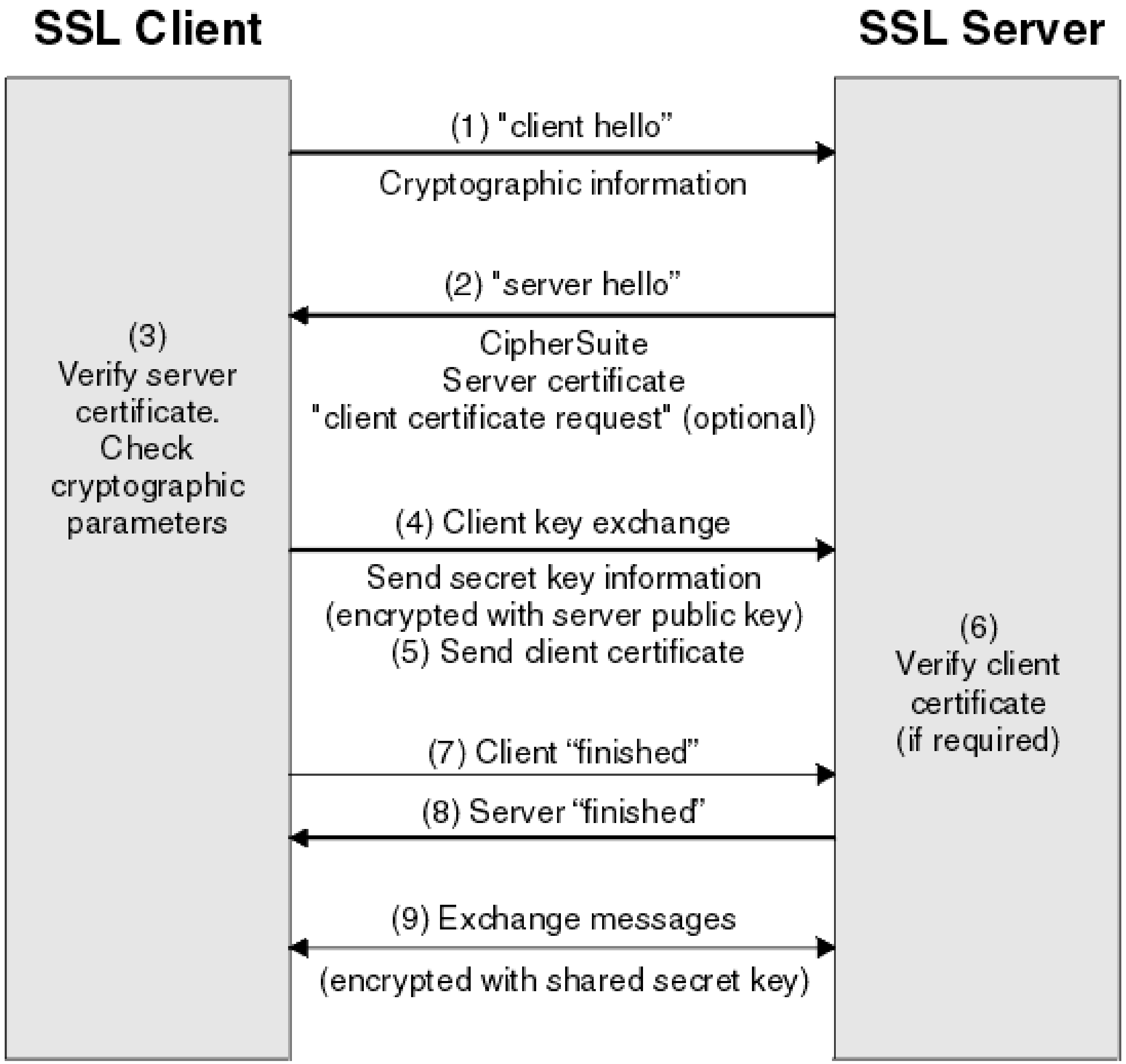
TLS (Transport Layer Security, RFC 2246): Objectives

Secure Communication Protocol over TCP/IP

- Evolved from the SSL V3 (Secure Sockets Layer) standard
- Manages secure sessions over TCP/IP, individual to each application
  - Initially designed for HTTP traffic
  - Currently used for many other types of traffic

Security mechanisms

- Confidentiality and Integrity of the communication between entities
  - Key distribution, Negotiation of ciphers, digests and other mechanisms
- Authentication of the intervenient entities
  - Servers, services, etc...
  - Clients
  - Both executed with asymmetric keys and X.509 certificates


SSL Client
SSL Server
(1) "client hello"
Cryptographic information
(2) "server hello"
CipherSuite
Server certificate
"client certificate request" (optional)
(3) Verify server certificate. Check cryptographic parameters
(4) Client key exchange
Send secret key information (encrypted with server public key)
(5) Send client certificate
(6) Verify client certificate (if required)
(7) Client "finished"
(8) Server "finished"
(9) Exchange messages
(encrypted with shared secret key)

TLS Ciphersuites

If a server supports a single algorithm, it not expected for all clients to also support it

- More powerful/limited, older/newer

The Ciphersuite concept allows the negotiation of mechanisms between client and server

- Both send their supported ciphersuites, and select one they both share
- TLS v1.3: O servido escolhe

Exemplo: ECDHE-RSA-AES128-GCM-SHA256

Format:

- Key negotiation algorithm: ECDHE
- Authentication algorithm: RSA
- Cifra algorithm, and cipher mode: AES-128 GCM
- Integrity control algorithm: SHA256

SSH (Secure Shell): Objectives

Manages secure console sessions over TCP/IP

- Initially designed to replace the telnet application/protocol
- Currently used in many other applications
  - Execution of remote commands in a secure manner (rsh/rexec)
  - Secure copy of contents from/to remote hosts (rcp)
  - Secure FTP (sftp)
  - Secure (Generic) communication tunnels (carry standard IP packets)

Security Mechanisms

- Confidentiality and integrity of the communications
  - Key distribution
- Authentication of the intervenient entities
  - Server / Hosts
  - Client users
  - Both achieved through several, and differentiated mechanisms

SSH: Authentication Mechanisms

Server: a pair of asymmetric keys

- Keys are distributed during the interaction
  - Not certified!
- Clients store the public keys from previous interactions
  - Key should be stored in some trusted environment
  - If the key changes the client is warned
    - e.g., server is reinstalled, key is regenerated, an attacker is hijacking the connection
    - Client can refuse to continue with the authentication process

Clients: authentication is configurable

- Default: username and password
- Other: username + private key
  - The public key MUST be pre-installed in the server
- Other: integration with PAM for alternative authentication mechanisms

SSH: Server Example

Long lived keys in /etc/ssh/
- Private: ssh_host_rsa_key
- Public: ssh_host_rsa_key.pub
  - Sent to users when they connect
  - No additional key management process (usage, validity, certification)

List of prime numbers
- /etc/sshd/moduli
- Used when establishing DH exchanges with clients

Can restrict specific users from connecting

Can interact with underlying authentication processes
- PAM: Pluggable Authentication Modules
- KRB: Kerberos
- GSSAPI: Generic Security Services Application Program Interface

SSH: Client Example

Per user information in ~/.ssh

- Both in the client and the server

Client:

- Keys for key based authentication
  - Private: id_ed25519
  - Public: id_ed25519.pub
- Config: Changes the behaviour to all or to specific servers
- known_hosts: Stores the public keys from all previous interactions

Server:

- authorized_keys: stores public keys for key based authentication

```
Reading configuration data /home/user/.ssh/config
Reading configuration data /etc/ssh/ssh_config
Connecting to server [127.0.0.1] port 22.
Connection established.
identity file /home/user/.ssh/id_ed25519 type 3
Local version string SSH-2.0-OpenSSH_7.9
Remote protocol version 2.0, remote software version OpenSSH_7.4p1 Debian-10+deb9u4
match: OpenSSH_7.4p1 Debian-10+deb9u4 pat OpenSSH_7.0*,OpenSSH_7.1*,OpenSSH_7.2*,OpenSSH_7.3*,OpenSSH_7.4*,OpenSSH_7.5*,OpenSSH_7.6*,OpenSSH_7.7* compat
0x04000002
Authenticating to server:22 as 'user'
SSH2_MSG_KEXINIT sent
SSH2_MSG_KEXINIT received
kex: algorithm: curve25519-sha256
kex: host key algorithm: ecdsa-sha2-nistp256
kex: server->client cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none
kex: client->server cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none
expecting SSH2_MSG_KEX_ECDH_REPLY
Server host key: ecdsa-sha2-nistp256 SHA256:GNK1+Z/XV/vYxuqqgrrZE45Gh5GqJeRPg6nFwrc+iYz
Host 'server' is known and matches the ECDSA host key.
Found key in /home/user/.ssh/known_hosts:2
rekey after 134217728 blocks
SSH2_MSG_NEWKEYS sent
expecting SSH2_MSG_NEWKEYS
SSH2_MSG_NEWKEYS received
rekey after 134217728 blocks
Will attempt key: /home/user/.ssh/id_ed25519 ED25519 SHA256:gtHwersg454erafrvsyerGdfadfSDgartagaeRG2fXZ
SSH2_MSG_EXT_INFO received
kex_input_ext_info: server-sig-algs=<ssh-ed25519,ssh-rsa,ssh-dss,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521>
SSH2_MSG_SERVICE_ACCEPT received
Authentications that can continue: publickey,password
Next authentication method: publickey
Offering public key: /home/user/.ssh/id_ed25519 ED25519 SHA256:gtHwersg454erafrvsyerGdfadfSDgartagaeRG2fXZ
Server accepts key: /home/user/.ssh/id_ed25519 ED25519 SHA256:gtHwersg454erafrvsyerGdfadfSDgartagaeRG2fXZ
Authentication succeeded (publickey).
Authenticated to server ([127.0.0.1]:22).
channel 0: new [client-session]
Requesting no-more-sessions@openssh.com
Entering interactive session.
pledge: network
client_input_global_request: rtype hostkeys-00@openssh.com want_reply 0
Requesting authentication agent forwarding.
```

Authentication in Systems

Devices and systems operate based on an identity
- With personal data is restricted to its owner
- Each system implements specific authentication processes

Validation against credentials/template
- Credentials/biometric template can be local
  - Frequently it is only local
- Can make use of secure execution mechanisms

Should provide offline authentication mechanisms
- Can support online mechanisms

Smartphones

Considered to be personal devices

- Frequently used to personally identify a person

Can exploit the existence of a SIM card or other HW

- Sold to an existing entity, Registered to an entity, Protected by a PIN code

Can use multiple authentication sources

- Passwords, PINs, Patterns, Biometrics

Supported by Trusted Environment

- Android: Trusty OS

Smartphones: Android

Uses a user-authenticated-gated keys

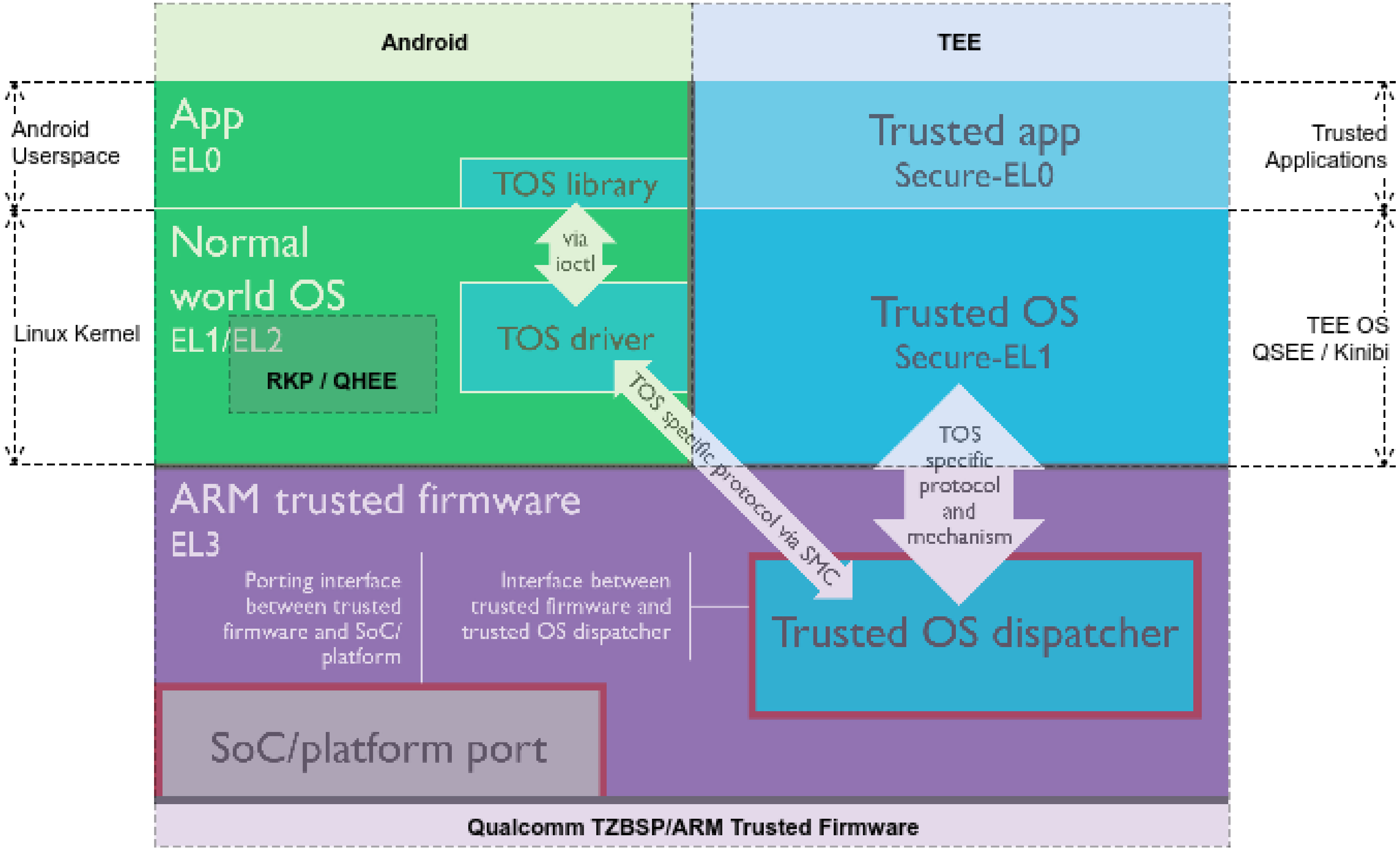
- Gate authenticates users to unlock keys
- Keystore stores keys in a protected environment

Security gates

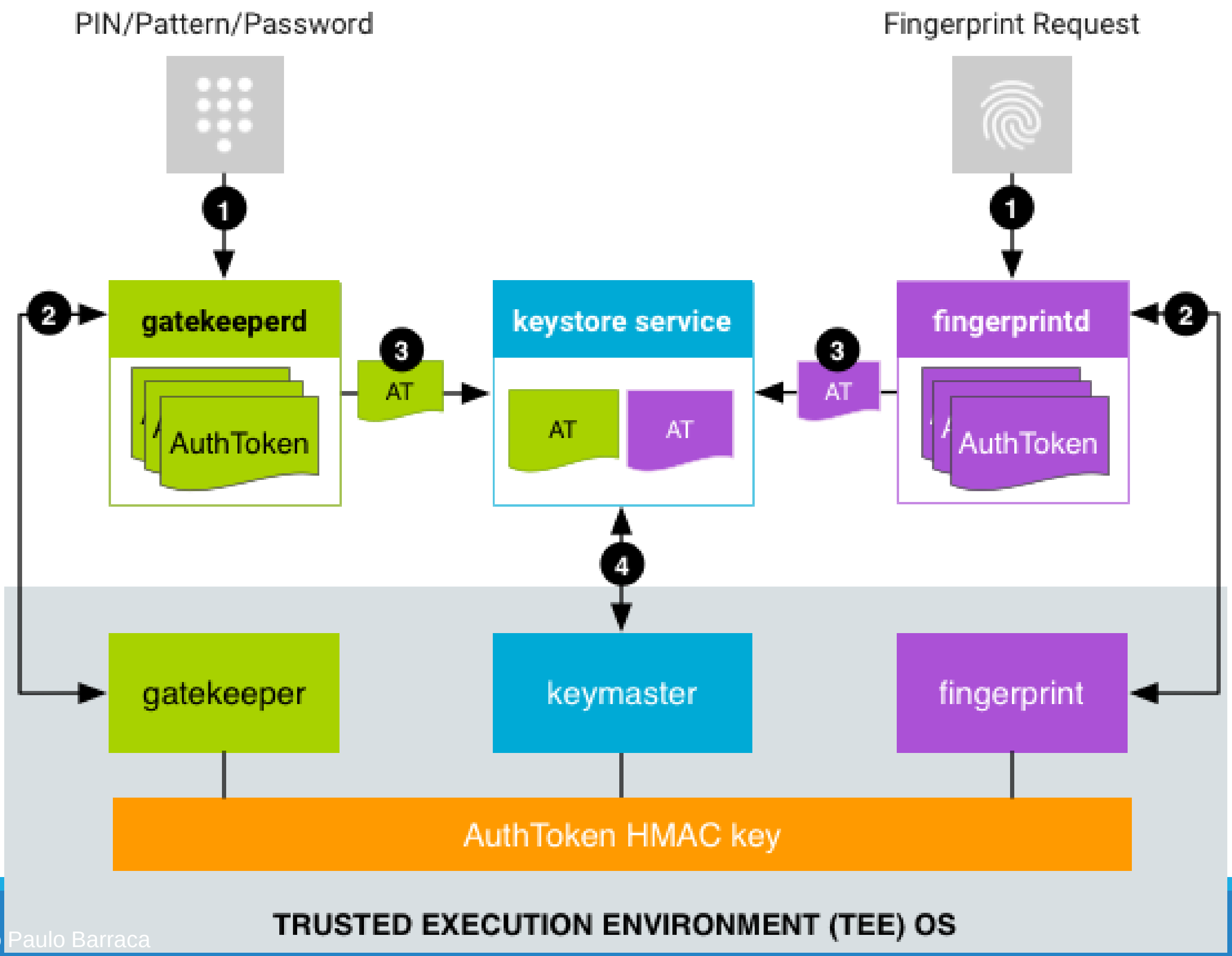
- Gatekeeper: for PINs/Passwords/Patterns
- Fingerprint: for fingerprints

PINs/Passwords/Patterns tied to an identity

- providing a pin unlocks its keys
- Secret keys tied to a user


Android
TEE
Android Userspace
App
EL0
TOS library
Trusted app
Secure-EL0
Trusted Applications
Linux Kernel
Normal world OS
EL1/EL2
RKP / QHEE
via ioctl
TOS driver
Trusted OS
Secure-EL1
TEE OS
QSEE / Kinibi
TOS specific protocol via SMC
TOS specific protocol and mechanism
ARM trusted firmware
EL3
Porting interface between trusted firmware and SoC/ platform
Interface between trusted firmware and trusted OS dispatcher
Trusted OS dispatcher
SoC/platform port
Qualcomm TZBSP/ARM Trusted Firmware

Smartphones: Android



Smartphones: Android Gatekeeper

Initial enrollment required
- Identity plus shared secret (PIN, Password, Pattern)
- 64bit random User Secure ID is generated and stored

Gatekeeper in the App Environment
- Sends SID + credentials to TEE
- Receives signed AuthToken
- Contacts keystore to obtain keys

Trusted Environment
- Validates credentials for SID
- Generates with valid AuthToken

Smartphones: AuthToken

Field	Type	Description
AuthToken Version	8 bits	Group tag for all fields.
Challenge	64 bits	A random integer to prevent replay attacks. Usually the ID of a requested crypto operation. Currently used by transactional fingerprint authorizations. If present, the AuthToken is valid only for crypto operations containing the same challenge.
User SID	64 bits	Non-repeating user identifier tied cryptographically to all keys associated with device authentication.
Authenticator ID (ASID)	64 bits	Identifier used to bind to a specific authenticator policy. All authenticators have their own value of ASID that they can change according to their own requirements.
Authenticator type	32 bits	Gatekeeper (0), or Fingerprint (1)
Timestamp	64 bits	Time (in ms) since the most recent system boot.
AuthToken HMAC (SHA-256)	256 bits	Keyed SHA-256 MAC of all fields except the HMAC field. Key is generated when booting and never leaves the TEE

Smartphones: Keymaster

Provides access to the keystore

- API based, not full RW access
- Replies to requests from authorized services (shared secret), having a valid (recent) AuthToken

Keymaster 1: Android 6

- Signing API (sign, verify, import keys)

Keymaster 2: Android 7

- Support for AES and HMAC
- Key Attestation: Certifies keys (origin, property, usages)
- Version Binding: ties keys to OS and TEE version, preventing downgrades

Keymaster 3: Android 8

- ID Attestation: Key device identifiers are stored as HMAC(HWKEY, IDn)

Keymaster 4: Android 9

- Embedded Secure Elements: allowing embedded “smartcards”

Android: Keymaster Key Attestation

Objective: Ensure keys are originated from the TEE, and are authentic

Other assurances:

- Generated by the current TEE (based on its ID)
  - ID=HMAC_SHA256(instante temporal || AppID || R, HBK)
    - R = a tag::RESET_SINCE_ID_ROTATION, HBK: a secret Hardware Backed Key

Call: attestKey(KeyToAttest, attestParams)

Result: A X.509 certificate

- Signed by a specific root certificate
- With an extension containing the result

Smartphones: Gatekeeper auth

PIN: Direct input of a digit based code
- Usually 4 digits but can be changed up to 16 digits
- Not related to the SIM PIN
- Vulnerable to attacks using sensors (gyro/accell)

Password: Direct input of a stream of characters
- Usually limited to 16 chars
- Less vulnerable to attacks using sensors (gyro/accell)

Pattern: Direct input of a pattern
- Potentially more secure than 4 digit PINS
- Stored as a unsalted SHA-1 digest
- Vulnerable to over-the-shoulder attacks, grease marks

Smartphones: Fingerprint

TEE stores a multi sample profile of a fingerprint

- always encrypted, even inside TEE
- associated to a SID
- Deleted if user is removed from device

Profile is obtained from sensor, validated in TEE

- Cannot be extracted
- Fingerprint is sent to TEE for validation

Security level varies with sensor implementation

- Several implementations

Fingerprint types: Optical

Sensor takes picture of finger

- Can use LEDs for illumination

Only a 2D image

- fooled by pictures, fingerprint models, latent prints

Present in first versions and entry level devices

An optical sensor.

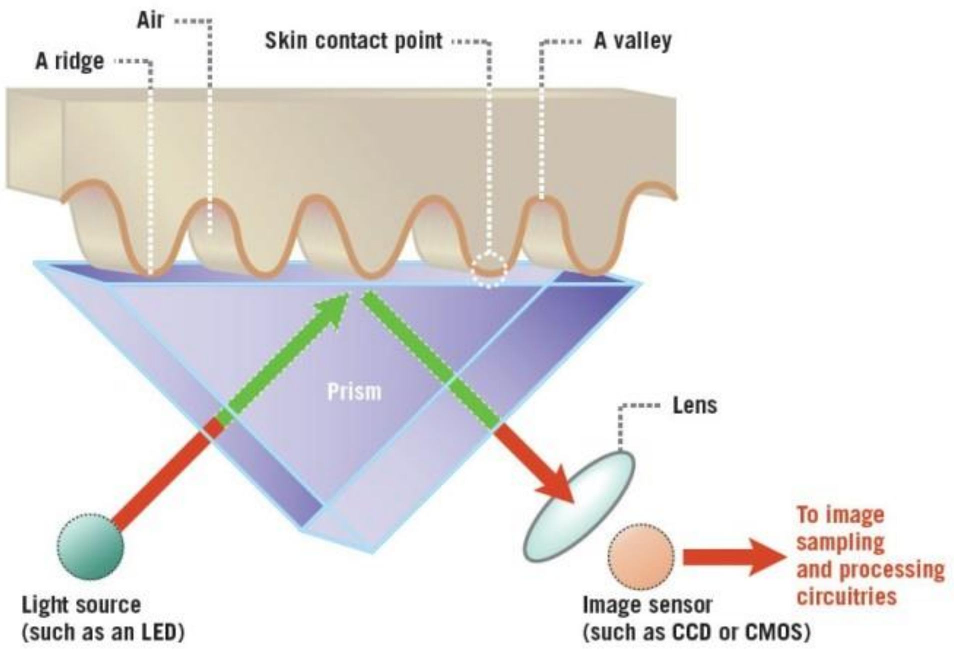

Air
A ridge
Skin contact point
A valley
Prism
Lens
To image sampling and processing circuitries
Light source (such as an LED)
Image sensor (such as CCD or CMOS)

Figure 2

Fingerprint types: Capacitive

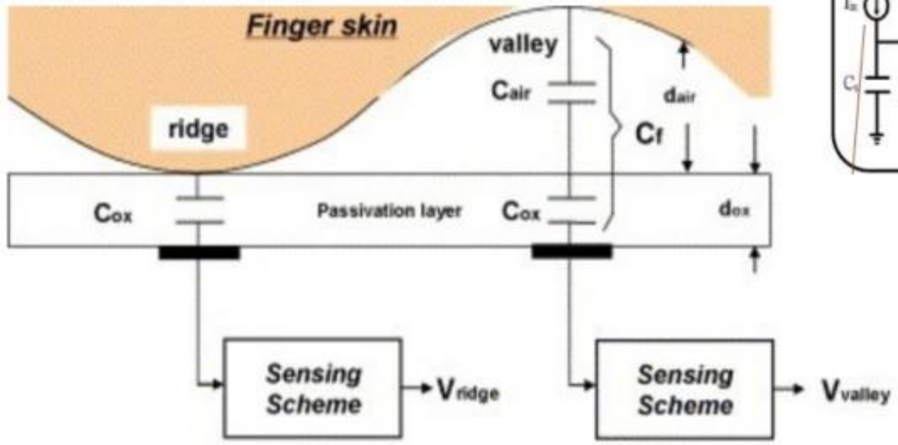
Sensor measures capacitance along the surface

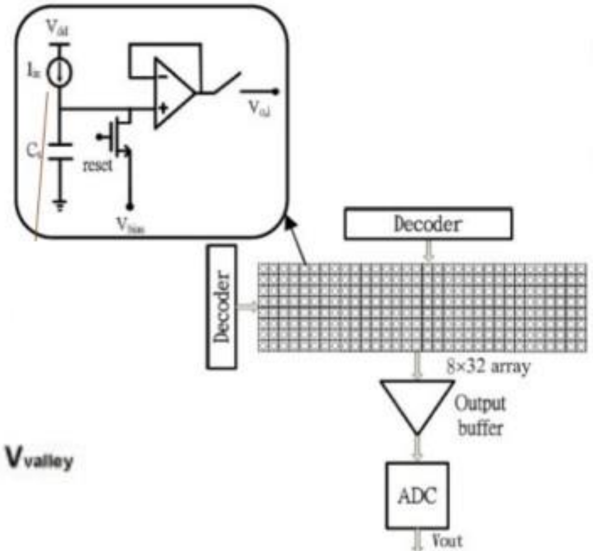
- Ridges and valleys (in sub-epithermal layers)
- Allows for Swipe implementations (cheaper versions)

Vulnerable to prosthetic (silicone) fingers

- With model from authenticated user

Interference from sweat, hand lotions, water





Fingerprint types: Ultrasonic

Ultrasound emitter and receiver

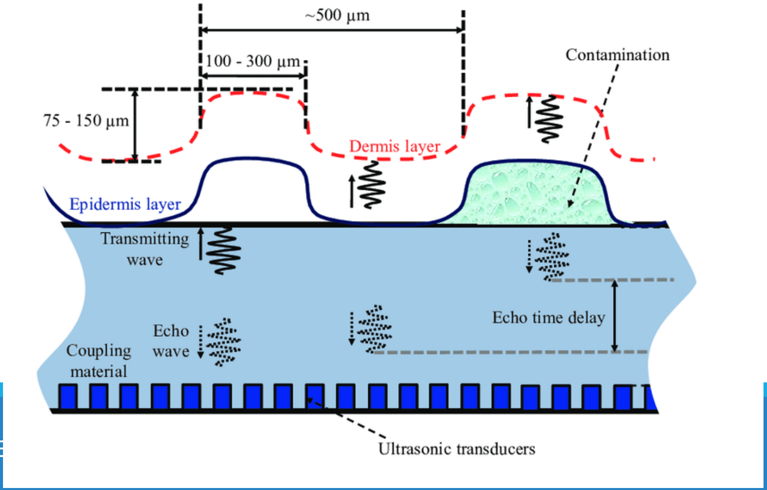
- Emitter: Emits pulse that is transmitted to the finger
- Received: listens for echos as sound encounters features

More difficult to circumvent and more resilient to surface material

- Echos penetrate through water, lotion, and bumps on features

Still possible…

- youtube/watch?v=hJ35ApLKpN4



Smartphones: Face Recognition

Objective: Match face against <u>trained</u> model

- Based on commonly available face recognition software

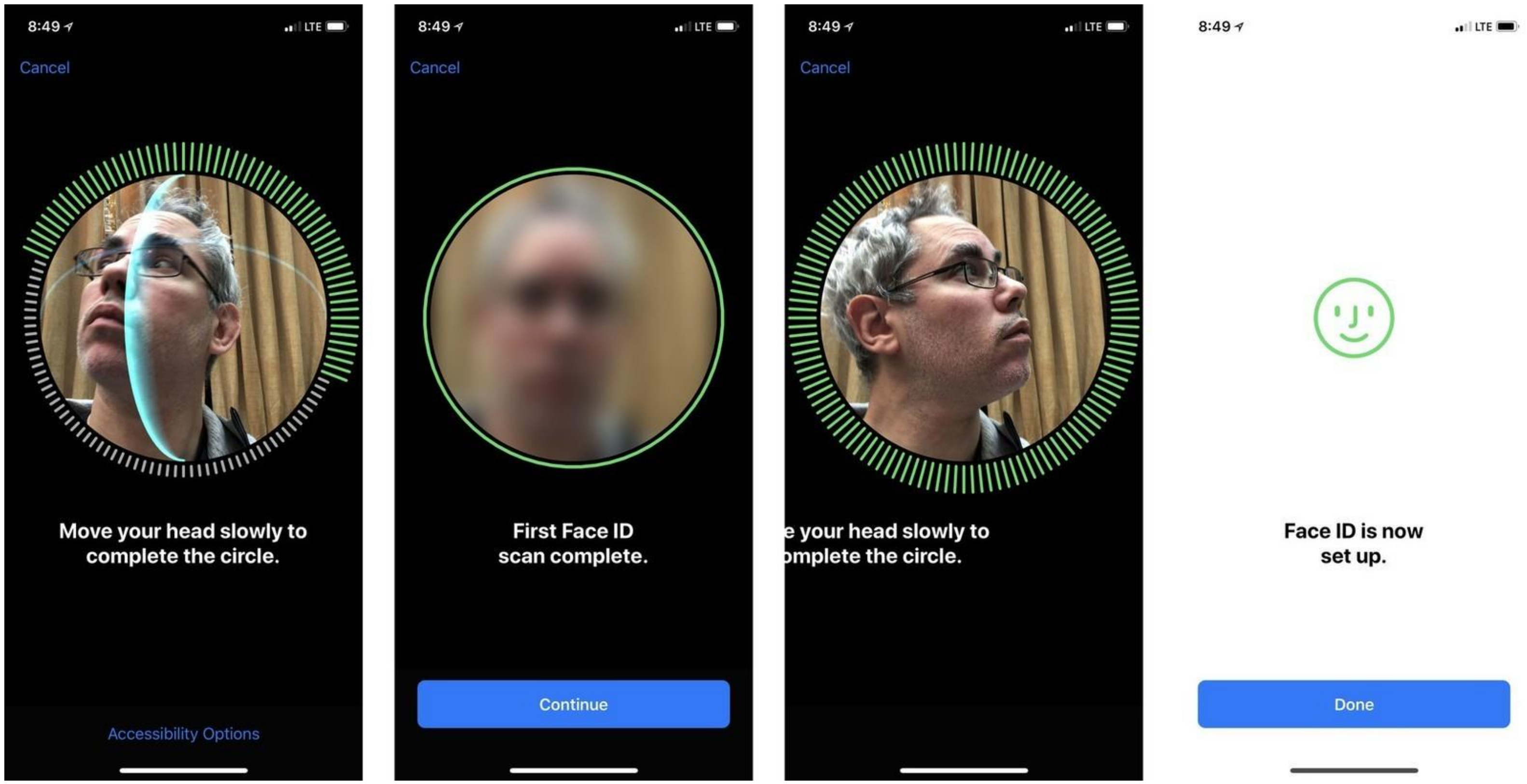
Requires initial enrollment to create train model

- Successful authentication can increase train data

Has some issues:

- Simple image can be fooled by a picture/movie/evil twin
- Not resilient to changes in lighting
- Not resilient to changes of the subject (glasses, beard)
- Not resilient to changes in posture

Smartphones: Face ID



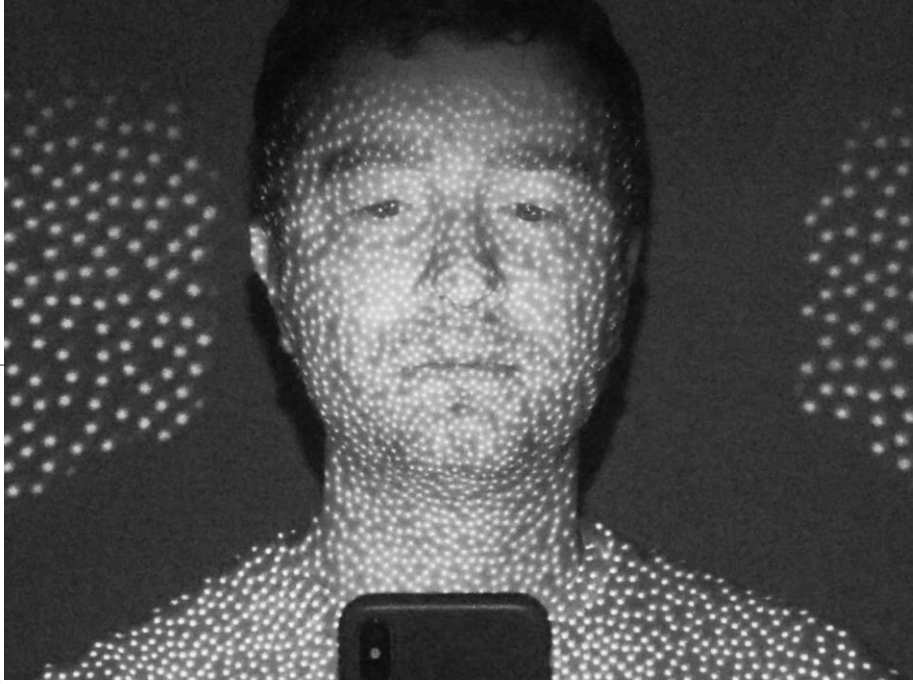
Move your head slowly to complete the circle.

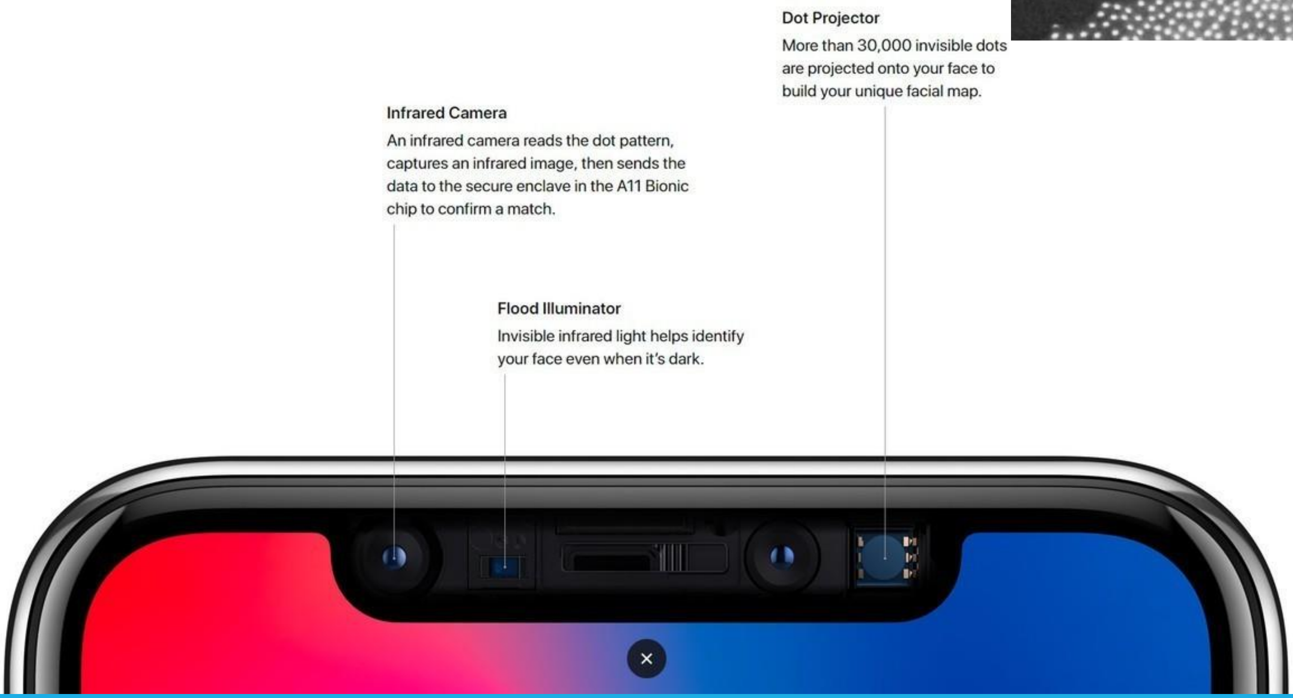
First Face ID scan complete.

e your head slowly to
omplete the circle.

Face ID is now set up.

Face ID




Dot Projector
More than 30,000 invisible dots are projected onto your face to build your unique facial map.
Infrared Camera
An infrared camera reads the dot pattern, captures an infrared image, then sends the data to the secure enclave in the A11 Bionic chip to confirm a match.
Flood Illuminator
Invisible infrared light helps identify your face even when it's dark.

Laptops

Laptops are considered as potentially shared devices

- Not really considered as individual devices
- May have some sensors/readers
- May have Trusted Platform Modules (TPM)

Authentication bound to underlying OS

- Simpler than smartphones
  - No SIM card
  - No TEE
  - Simpler biometric approach

No universal support for hardware backed key store

Laptops: Hardware support

Fingerprint sensors like in smartphones

- Swipe, discrete or in power button

Hardware for face recognition

- standard camera (standard in all laptops)
- infrared camera (more recent implementations)

Smartcard reader

- Allows use of traditional SmartCards (e.g., CC)
- More frequent in laptops for corporate environments

Can interact with other devices

- Smartphone, bracelet, Yubikey

OS: Windows

Supports a wide range of authentication methods

- PIN, Password, Biometrics, SmartCards, Tokens
- supports remote authentication (e.g., Active Directory)

Credentials stored in SAM (Security Account Manager)

- Optional: partially encrypted using SysKey
- trivial to remove a user password (delete SAM entry)
- Mapped to windows registry in HKLM/SAM

Since W Vista UAC enforces Access Control after authentication

- Vista launched in 2006
- UAC can still be disabled!

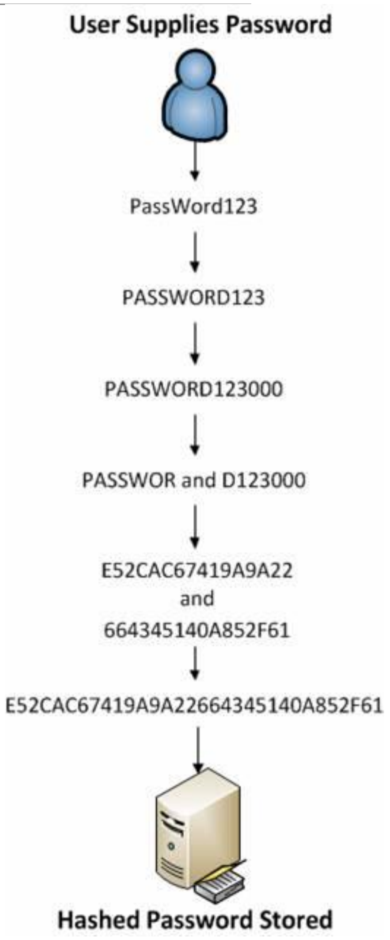
OS: Windows Passwords

Password: Direct validation against stored value
- Stored in c:\Windows\System32\Config\SAM
- Encrypted with Boot Key (SysKey)
- Complexity imposed by Admin Policy

LM Password Hash Up to W7
- Encrypts standard value (KGS!@#$%) using DES(password, standard)

NTLM Password Hash
- Non Salted MD4(Password)
- Same password -> same hash

Validation:
- Request username and password
- Calculates hash, compares the result with stored value



OS: Windows PIN

Backed by a Trusted Platform Module (TPM)

- Similar to TEE, provides secure environment with storage
- Can guarantee hardware tamper free state

PIN unlocks TPM which allows access to keys

- repeated incorrect attempts will lock TPM
- cannot be extracted (bound to device)

OS: Windows Hello

Uses Visible Light + IR cameras to obtain 3D image

- Can have LED for flood illumination
- IR camera adds resilience to lighting changes
- Two cameras introduce 3D depth data (from the parallax)
- PIN is mandatory as backup

Vulnerable

- to 3d printed face?
- to IR sensitive print
- to standard print in earlier W10

OS: Linux

Supports a wide range of authentication methods

- PIN, Password, Biometrics, SmartCards, Tokens
- supports remote authentication (e.g., Active Directory)

Pluggable Authentication Modules allows per app authentication policies/mechanisms

- without modification to applications
- e.g: SmartCards, OTP, Kerberos, LDAP, Databases, Network Location, etc..

Standard Credentials stored in /etc/shadow

- not encrypted
- Alternate authentication methods may use other storage (e.g., TPM, SmartCard, Database)

OS: Linux Passwords

User account info in /etc/passwd

- username, user id, shell...

Credentials stored in /etc/shadow

- only readable by root, transformed using a salted digest

Validation:

- obtain credential from user
- access shadow: verify hash used and obtain salt
- calculate hash(salt + password) for N rounds (default is 5000)
- compare result obtained

Entry:
user:$6$kZ2HbBT/C8MxFlN1$YWNjZDczOWVmNWNmNjBiYmRlNjBmYWUxZTc4YTJmM2FjZDVmNGU3MmM3MjI2YzZkYzI2YjRlMDU4:17716:0:99999:7:::

Meaning: username:$ hash used $ salt $ password hash: ... dates and validity

SSO: Single Sign On

Unique, centralized authentication for a set of federated services

- The identity of a client, upon authentication, is given to all federated services
- The identity attributes given to each service may vary
- The authenticator is called Identity Provider (IdP)

Examples

- SSO authentication at UA
  - Performed by a central IdP (idp.ua.pt)
  - The identity attributes are securely conveyed to the service accessed by the user

SSO: Single Sign On

Trusted third parties (TTP) used for authentication

- But often combined with other related functionality
- E.g. Google, Facebook

AAA services

- Authentication, Authorization and Accounting
- e.g. RADIUS and DIAMETER

SSO: Single Sign On

Advantages:

- Can reuse same credentials over multiple systems/services
- Single secure repository for credentials
  - More difficult to steal credentials when used in many services
- Can implement restrictions to services/systems

Disadvantages:

- Requires additional servers
- Single point of failure: without authentication systems, no one will be authenticated
  - Important to also deploy local credentials for admins
- Introduces delays in the authentication process

SSO: Single Sign On

Requires software that "injects" remote users into local system

- Windows: Remote users not available in SAM
- Linux: Remote users not available in /etc/passwd
- Must cache data to enable large number of validations (e.g., ls)

May provide further information to be used as user profile

- Type of user (student, professor, admin)
- email, home, other preferences

Systems that make use of SSO need to be provisioned

- And sometimes, specifically authorized

SSO: LDAP
Lightweight Directory Access Protocol

Protocol to keep distributed directory information

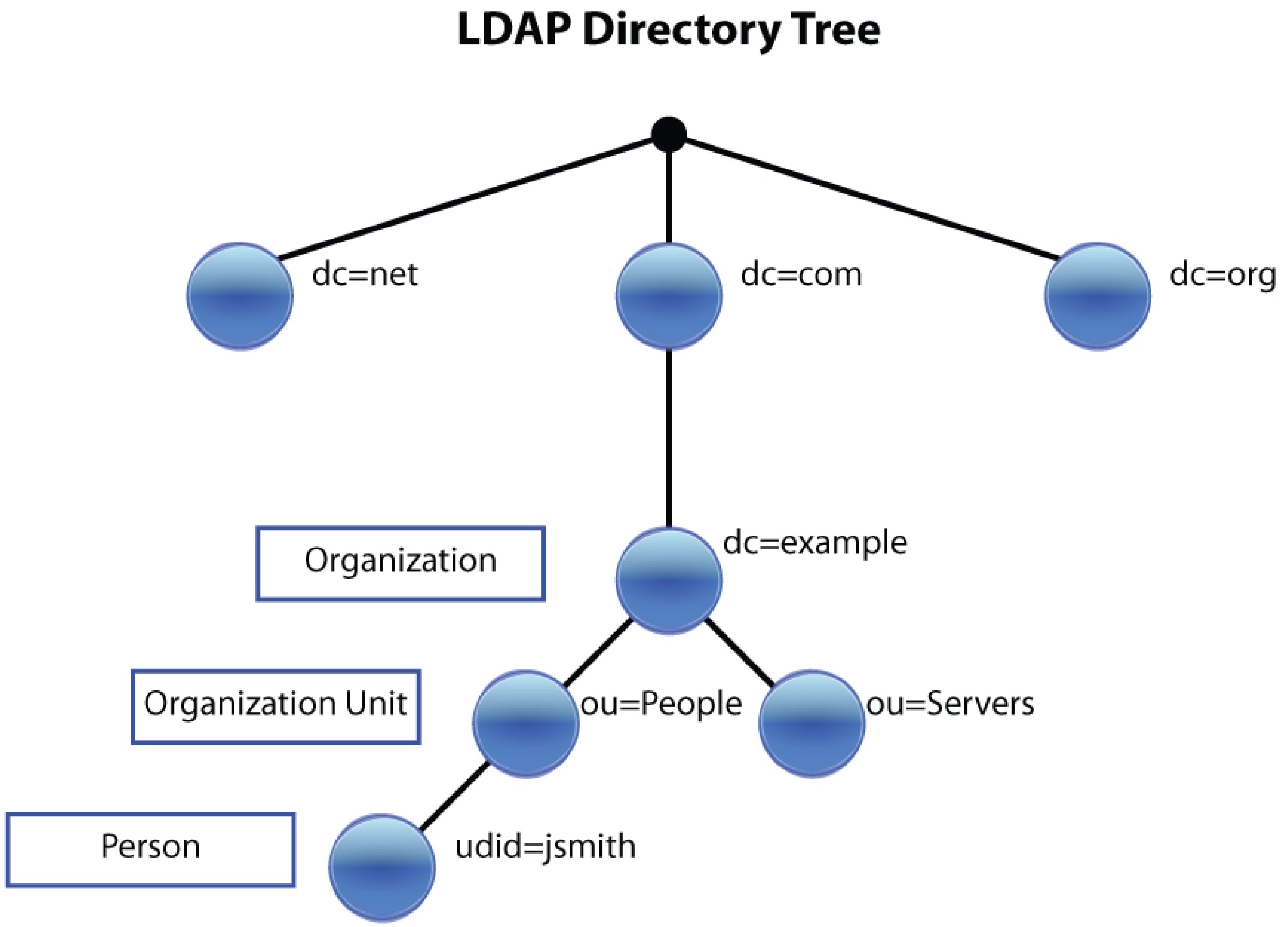
- Directory keeps hierarchical information about users, systems and services
  - E.g., address book, user profile
- Information is organized in a tree: dn=user,ou=deti,dc=ua, dc=pt
  - DC: Domain Component, OU: Organizational Unit, DN: Distinguished Name
- Each record obeys to a specified composition of individual schemas

Access to LDAP can be anonymous or authenticated

- Anonymous information: general contacts and configurations
- Authenticated (Bind): Specific profile info

LDAP Bind: credentials are user path and password

- Support for different authentication methods: PLAIN, SASL, Certificates
- Supports same username in different domains
  - **dn=usera,ou=deti,dc=ua,dc=pt** vs **dn=userb,ou=deti,dc=ua,dc=pt**


LDAP Directory Tree
dc=net
dc=com
dc=org
Organization
dc=example
Organization Unit
ou=People
ou=Servers
Person
udid=jsmith

SSO: Kerberos

Authentication protocol for usage in networked environments

- Based on the notion of Tickets with limited validity
- Default process for Microsoft Active Directory (and CodeUA)

Supports mutual authentication

- Actually, the authenticator will send the password to the client!

Four Key Entities

- Client: Wishes to access a service
- Service Server (SS): Provides a service the user wants to access
- Ticket Granting Server (TGS): Provides access to services
- Authentication Server (AS): Provides access to the TGS for each user

Key Distribution Center: AS + TGS (+database)

SSO: Kerberos: Client Authentication

1: Client password is transformed (e.g. hash)

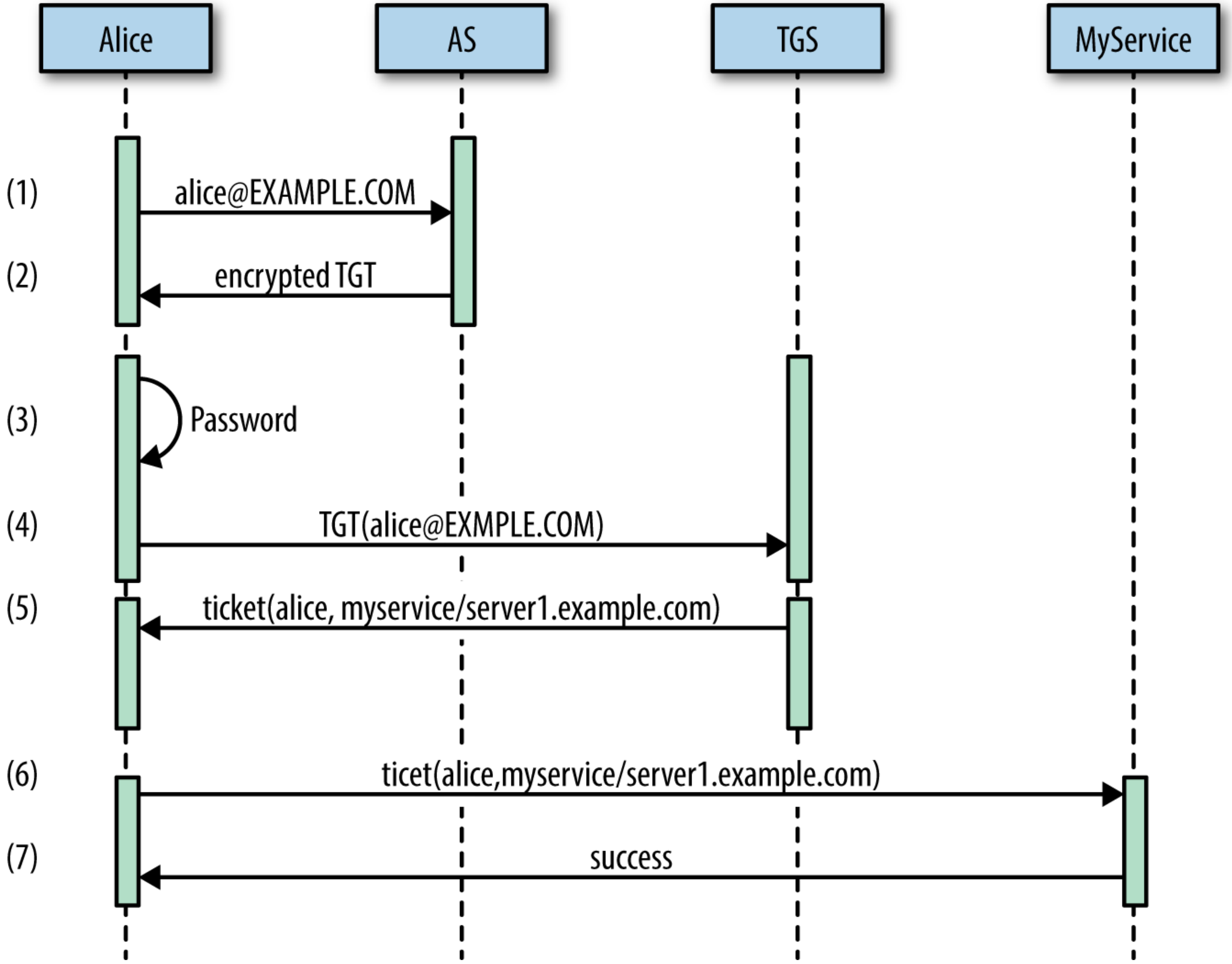
2: Client sends authentication request to AS with ClientID

3: AS replies with 2 messages:

- A: E_{user_key}(Client/TGS Session Key)
- B: E_{tgs_key}(TGT)
  - Ticket Granting Ticket = Client, client network address, validity, Client/TGS Session Key

4: User uses its key to decrypt A

- if password equals the one stored in AS he has access to TGS Session Key
- He can request Authorization to access the Service


Alice
AS
TGS
MyService
(1)
alice@EXAMPLE.COM
(2)
encrypted TGT
(3)
Password
(4)
TGT(alice@EXMPLE.COM)
(5)
ticket(alice, myservice/server1.example.com)
(6)
ticet(alice,myservice/server1.example.com)
(7)
success