Politecnico di Torino 1859

Authentication techniques, protocols, and architectures

Diana Gratiela Berbecaru
diana.berbecaru@polito.it

Politecnico di Torino
Dip. Automatica e Informatica

AY. 2023 - 2024

Acknowledgment

- **Slides content has been prepared by Prof. Antonio Lioy for the course Information Systems Security (2005 - 2022)**
  - minor modifications applied
- **… so this set of slides is entirely compatible with the course of the previous year(s)**

Definitions of authentication

- **RFC-4949 (Internet security glossary)**
  - ‘the process of verifying a claim that a system entity or system resource has a certain attribute value’
- **whatis.com:**
  - ‘the process of determining whether someone or something is who or what it is declared to be’
- **NIST IR 7298 (Glossary of Key Information Security Terms)**
  - ‘verifying the identity of a user, process, or device, often as a prerequisite to allowing access to resources in an information system’

Definitions of authentication

- **authentication of an ‘actor’**
  - human being (interacting via software running on hardware), typically called **user authentication**
  - software component
  - hardware element (interacting via software)
- **shorthand: authN (or also authC)**
- **different from authorization (authZ) but related**

Authentication factors

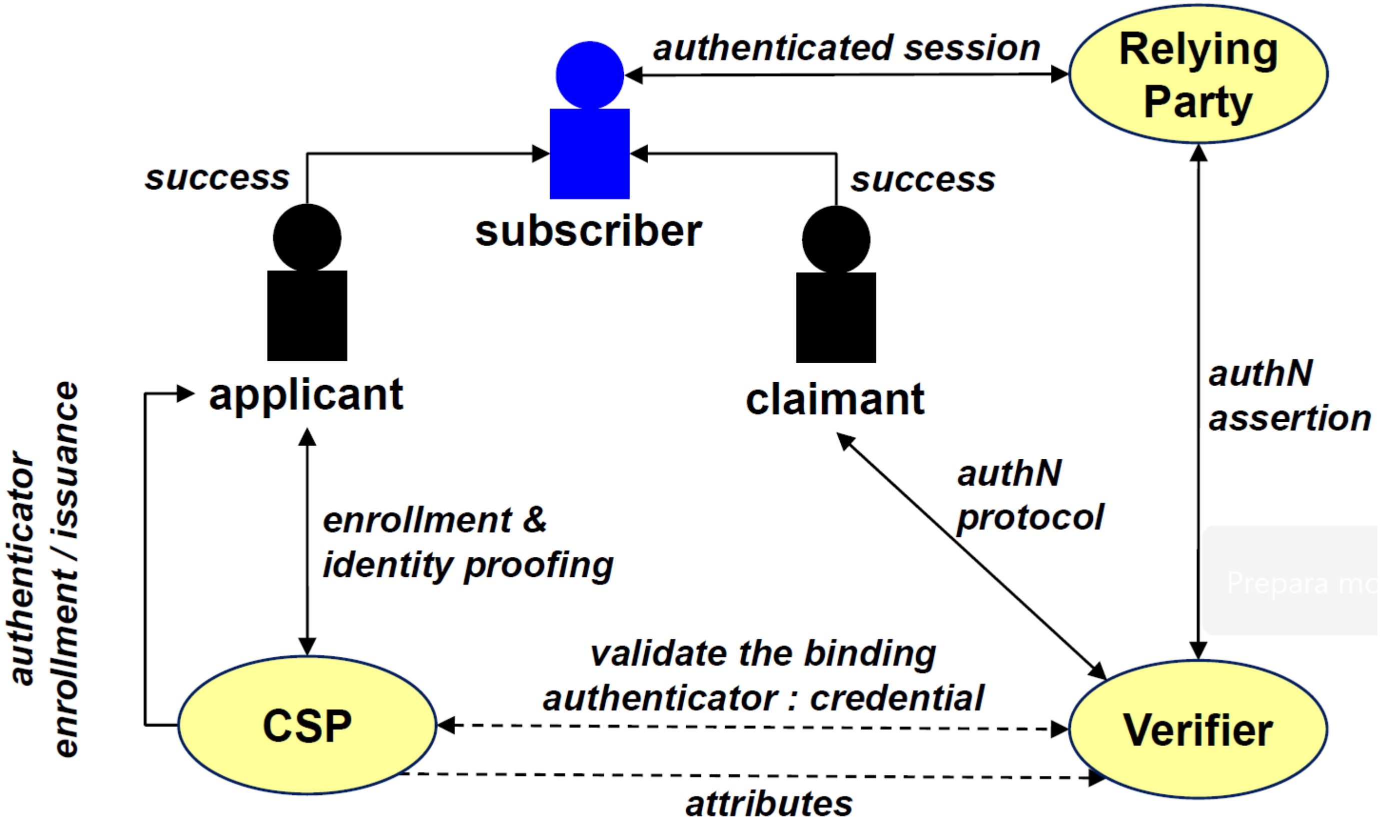
- **authentication can be based on different factors (1/2/3-factors authentication):**
  - knowledge = something only the user knows, e.g. a static password, code, personal identification number (PIN)
  - ownership = something only the user possesses (often called an ‘authenticator’), e.g. token, smart card, smartphone
  - inherence = something the user is, e.g. a biometric characteristics (such as a fingerprint)
- **different mechanisms can be combined (=multi-factor authentication, MFA)**
- **consider application not only to human users but also to processed and devices**

Opens67$$h

Authentication factors: risks

- **knowledge (e.g. password)**
  - risks = storage and demonstration/transmission
- **ownership (e.g. smartphone)**
  - risks = authenticator itself, theft, cloning, unauthorized usage
- **inherence (e.g. biometrics)**
  - risks = counterfeiting and privacy
  - cannot be replaced when ‘compromised’ (big problem!)
  - use it only for local authentication, as a mechanism to unlock a secret or a device

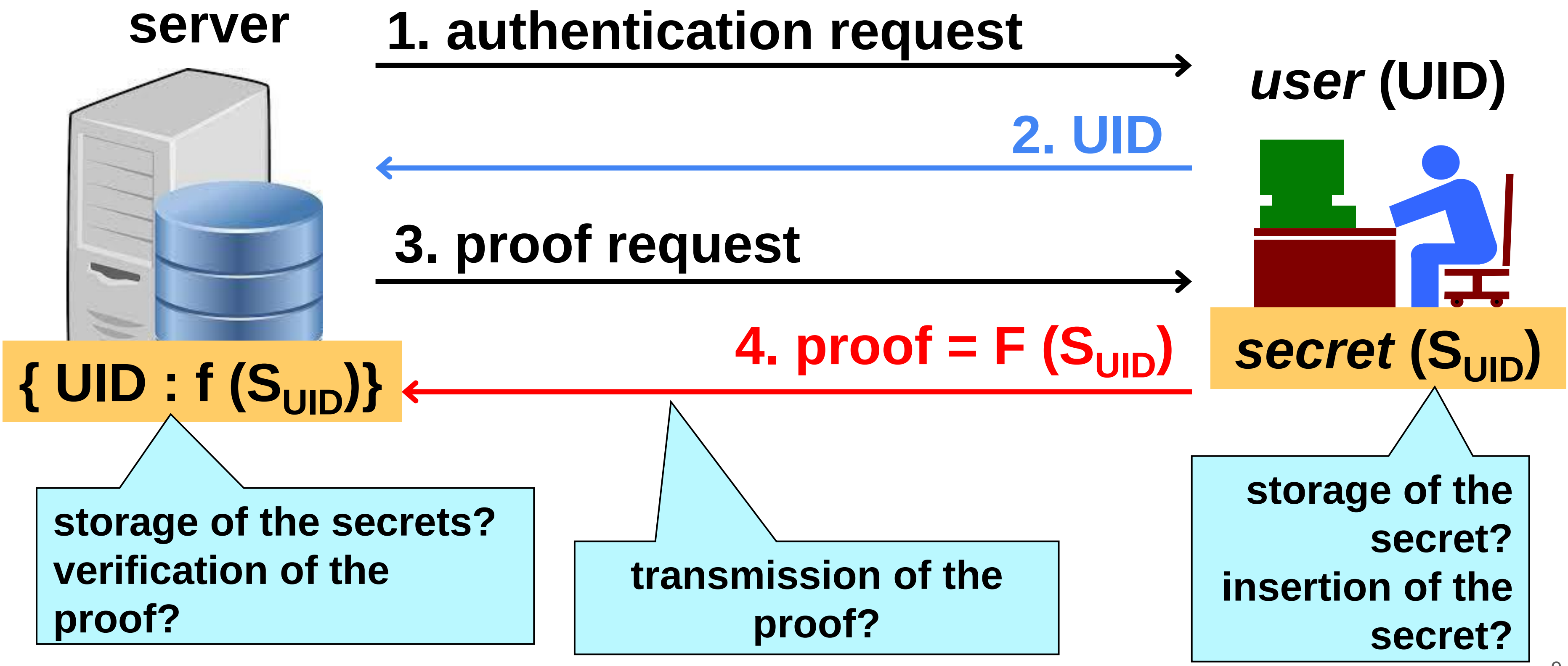
Digital authentication model (NIST SP800.63B)



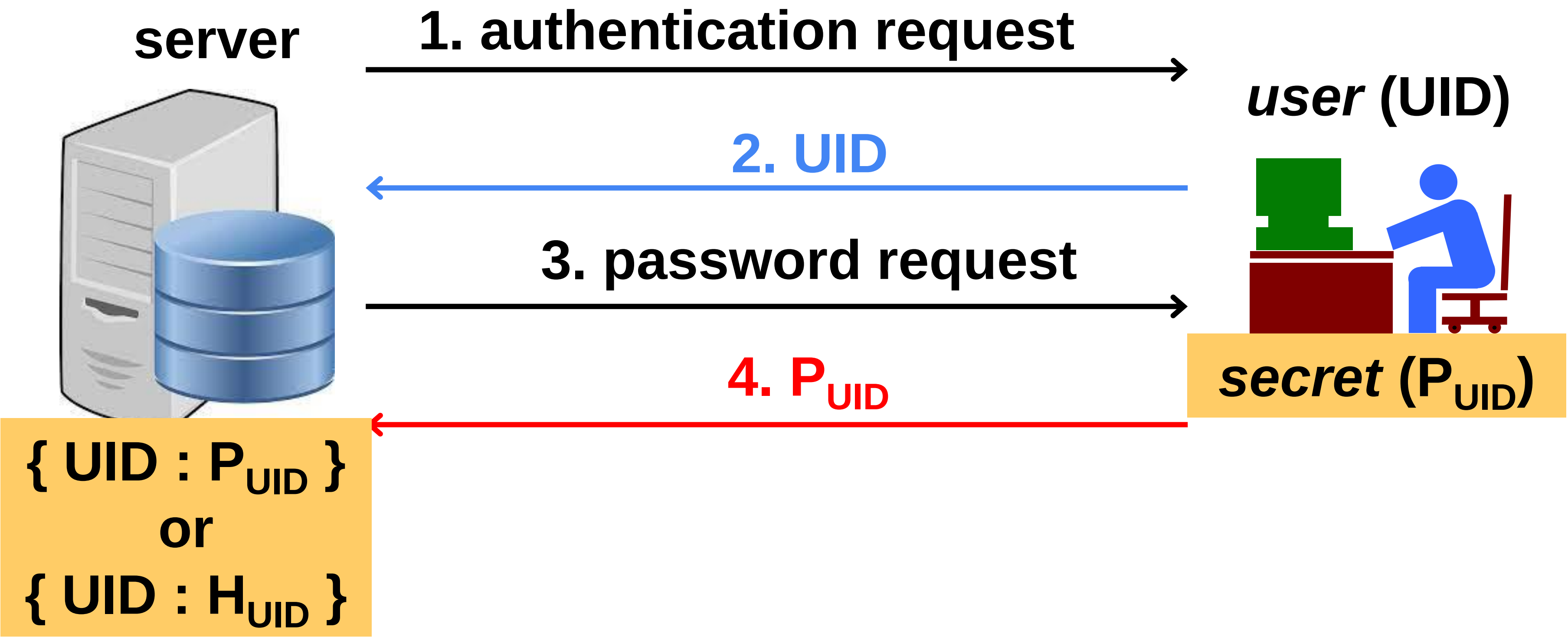
Digital authentication: entities

- **CSP (Credential Service Provider)**
  - will issue or enroll user credential and authenticator
  - verify and store associated attributes
- **Verifier**
  - executes an authN protocol to verify possess of a valid authenticator and credential
- **Relying Party (RP)**
  - will request/receive an authN assertion from the verifier to assess user identity (and attributes)
- **credential binds an authenticator to the subscriber, via an ID**
  - e.g. an X.509 certificate
- **these roles may be separated or collapsed together**

Generic authentication protocol



Password (reusable)



Password-based authentication

- **secret = the user password**
- **case #1: f = I (the identity function)**
  - (client) create and transmit proof
    - i.e. proof = password (cleartext!)
  - (server) verify the proof:
    - server knows all passwords in cleartext (!)
    - access control: proof = password ?
  - subject to sniffing and replay attacks

Password-based authentication (II)

- **secret = the user password**
- **case #2: f = one-way hash (that is a digest)**
  - (client) create and transmit proof
    - i.e. proof = $h(P_{UID})$
  - (server) verify the proof:
    - knows the passwords' digests, HUID
    - access control: $h(P_{UID})$ = HUID ?
  - subject to replay attacks (and dictionary attack)

Password-based authentication (III)

- **pro:**
  - simple for the user
  - … only if she has to remember just one password!
- **cons:**
  - user-side password storage
    - post-it!
    - client-side password manager or wallet
  - guessable password (my son’s name!)
  - server-side password storage – the server must know the password in cleartext or an unprotected digest of it (dictionary attack)

Other password problems (I)

- **pwd sniffing**
- **pwd DB attacks (if DB contains plaintext or obfuscated pwd)**
- **pwd guessing (very dangerous if it can be done offline, e.g. against a list of pwd hashes)**
- **pwd enumeration**
  - if pwd limited in length or character type
  - if authN protocol does not block repeated failures
- **pwd duplication**
  - using the pwd for one service against another one, due to user pwd reuse

Other password problems (II)

- **cryptography ageing**
  - flexibility on algorithms due to new attacks and more computing power
- **pwd capture via server spoofing and phishing**
- **MITM attacks**

Password best practice

- **suggestions to reduce the associated risks:**
  - alphabetic characters (uppercase + lowercase) + digits + special characters
  - long (at least 8 characters)
  - never use dictionary words
  - frequently changed (but not too frequently!)
  - don’t use them (?)
    - use of at least one password (or PIN, or access code, or ...) is unavoidable for usability reasons
    - sometimes used together with biometric techniques
      - care! If biometric system fails then the authN strength depends on the password

Storing the password: server-side

- **server-side**
  - NEVER in cleartext!
  - encrypted password? then the server must know the key in cleartext …
  - store a digest of the password
  - … but beware of the “dictionary” attack
  - … that can be made faster by a “rainbow table”
  - we must therefore insert an unexpected variation, usually named “salt”

Storing the password: client-side

- **client-side**
  - should be in the user’s head …
  - too many passwords … use an encrypted file (or a ‘password wallet’ or a ‘password manager’)

The “dictionary” attack

- **hypothesis:**
  - unknown password, pwd
  - known hash algorithm, h()
  - known hash value or the password, HP = h(pwd)
- **attack (pre-computation):**
  - for (each Word in Dictionary) do
    store (DB, Word, hash(Word))
- **attack:**
  - attacker gets HP = the hash value of a (unknown) password
  - w = lookup (DB, HP)
  - if (success) then write (“pwd = ”, w)
    else write (“pwd not in my dictionary”)

The “dictionary” attack - notes

- **in ‘dictionary attack’ the pre-computation is what makes the attack successfull**
  - the attacker must have the pre-computed DB
- **starting the pre-computation after discovering HP (e.g. read it from a file, or sniff it from network) could take more time than the password lifetime**

Using the salt in storing passwords

- **for each user UID:**
  - create / ask the pwd
  - generate a $salt_{UID}$
    - different for each user
    - random (unpredictable)
    - long (increased dictionary complexity)
    - should contain rarely used or control characters
  - compute HP_{UID} = hash (pwd || $salt_{UID}$)
- **store the triples { UID, HP_{UID}, $salt_{UID}$ }**
- **additional benefit: we have different HP_{UID} for users having the same pwd**

Using the salt in storing passwords - notes

- **makes the dictionary attack nearly impossible**
  - including those based on rainbow tables (a space-time trade-off technique to enable exhaustive search for a character set)

Example: passwords in Linux

- **originally stored in /etc/passwd, hashed with a DES-based hash function named crypt()**
- **since /etc/passwd needs to be world-readable (contains usernames, UID, GID, home, shell, …) passwords have been moved to /etc/shadow readable only by system processes**
- **passwords are stored in the following form – see crypt(5):**
  - idsalt$hashedpwd
  - different hash functions used depending on ID, for example:
    - 1=MD5,…,5=SHA-256,6=SHA-512,…
  - if idsalt is absent then the old DES-based hash is used (with 12-bits salt, pwd truncated to 8 characters) – danger!

The Linkedin attack

- **June 2012, copied 6.5 M passwords from Linkedin**
  - …unsalted, plain SHA-1 hash!!!
- **crowdsourcing used for cooperative password cracking**
  - at least 236,578 passwords found (before ban of the site publishing the password hashes)

- **Note: nearly simultaneous problem with the discovery that Linkedin app for iPad/iPhone was sending in clear sensible data (not relevant to Linkedin!)**

Strong (peer) authN

- **‘strong authN’ often requested in specifications**
- **…but never formally defined (or defined in too many different ways, which is useless)**

Strong authN: European Central Bank definition

- **strong customer authN is a procedure based on the use of two or more of knowledge, ownership, and inherence**
- **the elements selected must be mutually independent, i.e. the breach of one does not compromise the other(s)**
  - remember separation of privilege security principle (!)
- **at least one element should be non-reusable and non-replicable (except for inherence), and not capable of being surreptitiously stolen via the Internet**
- **the strong authentication procedure should be designed in such a way as to protect the confidentiality of the authentication data**

Strong authN: PCI-DSS definition

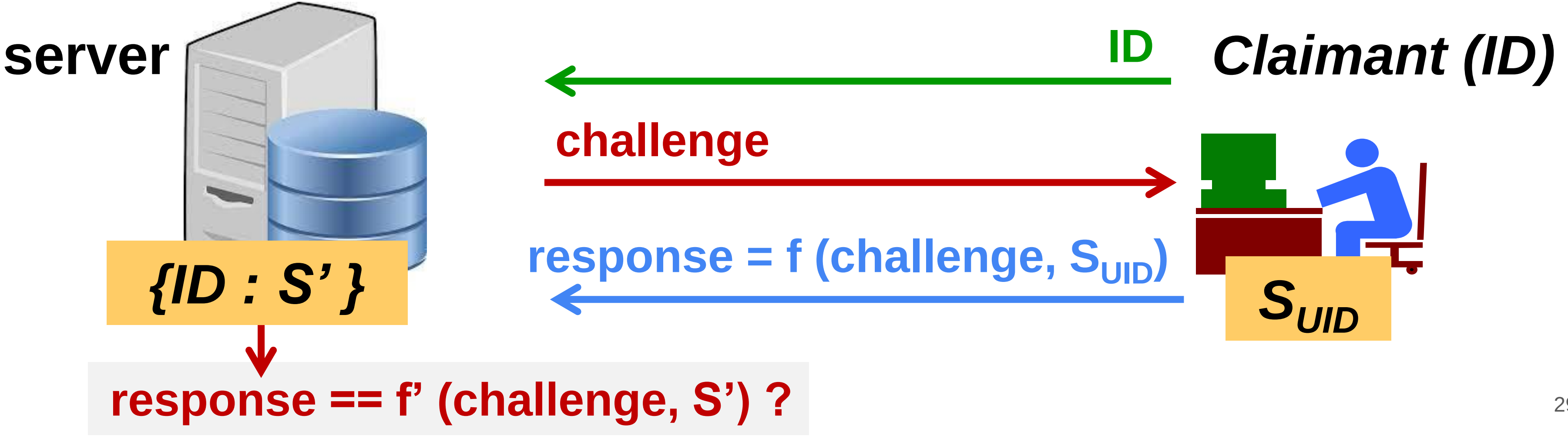
- **v3.2 requires multi-factor authentication (MFA) for access into the cardholder data enviornment (CDE)**
  - from trusted or untrusted network
  - by administrators
  - exception: direct console access (physical security)
- **… and for remote access**
  - from untrusted network
  - by users and third-parties (e.g. maintenance)
- **best practice until 2018/01, compulsory afterwards**
- **MFA is *not* twice the same factor (e.g. two passwords)**

Strong authN: other definitions

- **Handbook of Applied Cryptography**
  - a cryptographic challenge-response identification protocol
- **more in general**
  - technique resisting to a well-known set of attacks
- **conclusion:**
  - an authN technique can be regarded as strong or weak depending on the attack model
    - e.g. users of Internet banking > European Central Bank definition
    - e.g. employees of PSP > PCI-DSS definition
- **watch out for you specific application field (and risks)**

Challenge-response authentication (CRA)

- a challenge is sent to the Claimant ...
- ... who replies with a response computed using some secret knowledge (S_{UID}), a function f, and the challenge
- the server compares the response with a solution computed via data associated to the user (S’)



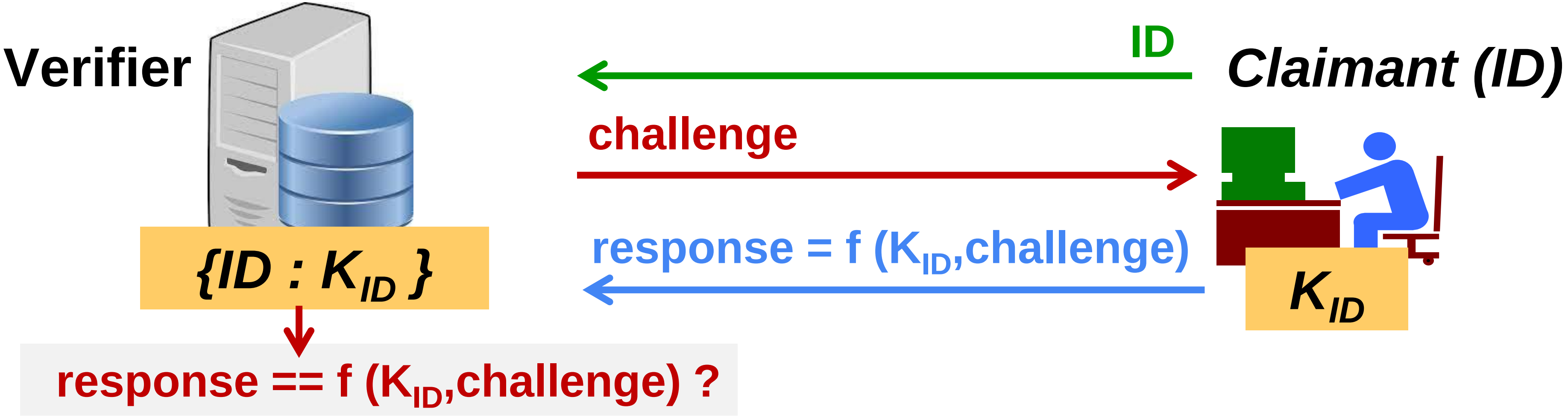
CRA: general issues

- **the challenge must be non-repeatable to avoid replay attacks**
  - usually the challenge is a (random) nonce
- **the function f must be non-invertible**
  - otherwise a listener can record the traffic and easily find the secret knowledge (S_{UID})

$$\boxed{S_{UID} = f^{-1}(\text{response}, \text{challenge})}$$

Symmetric challenge-response systems

- a challenge is sent to the Claimant ...
- ... who replies with a response computed using a symmetric function f, some shared secret key (K_{ID}), and the challenge
- the server compares the response with a solution computed via the secret key associated to the user (K_{ID})

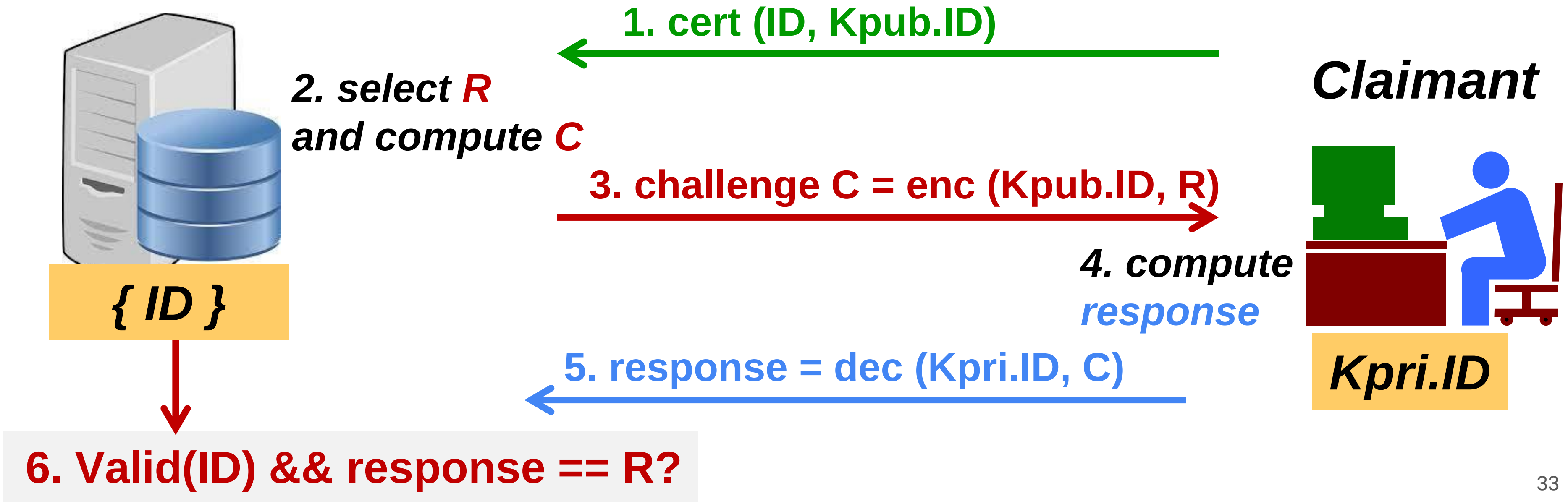


Symmetric CRA: general issues

- **the easiest implementation uses a hash function (faster than encryption) for the function f**
  - sha1 (deprecated), sha2 (recommended), sha3 (future)
- **K_{ID} must be known in cleartext to the verifier**
  - attacks against the {ID: K_{ID}} table at the verifier
- **… but solutions exists, e.g. SCRAM (Salted CRA Mechanism) solves this problem by using hashed passwords at the verifier**

(Asymmetric) challenge-response systems

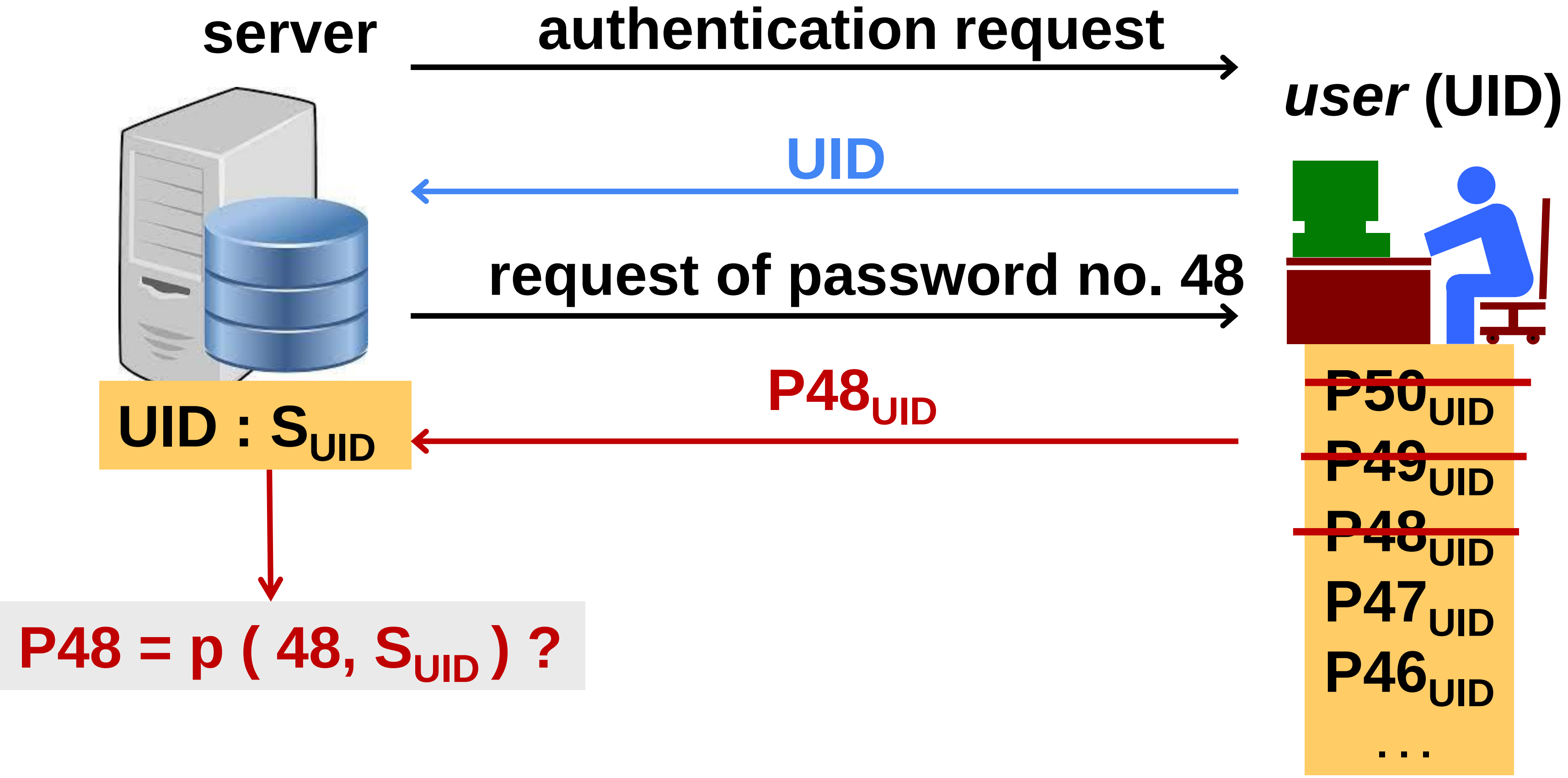
- a random nonce R is encrypted with the claimant's public key
- the claimant replies by calculating and sending response (thanks to knowledge of the corresponding private key, Kpri.ID)



Asymmetric CRA: analysis

- **advantages**
  - the strongest authentication mechanism
  - does not require secret (key) storage at the verifier
  - implemented for peer authentication (client and server) in IPsec, SSH, and TLS
  - cornerstone for user authentication in FIDO
- **disadvantages**
  - slow
  - if designed inaccurately may lead to an involuntary signature by the Claimant
  - PKI issues (trusted root, revocation). avoidable if the verifier stores Kpub.ID (moves equivalent PKI effort to the verifier)

One-time password (OTP)

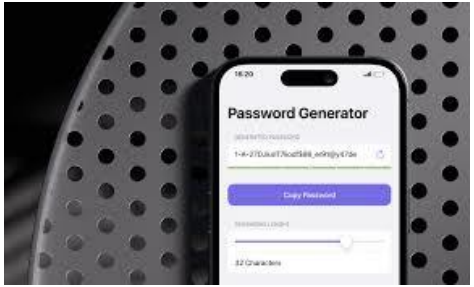


One-Time Password (OTP)

- **password valid only for one run of the authentication protocol**
  - next run requires another password
- **immune to sniffing**
- **subject to MITM (needs Verifier authentication)**
- **difficult provisioning to the subscribers**
  - lots of passwords
  - password exhaustion
- **difficult password insertion**
  - random characters to avoid guessing

OTP provisioning to the users

- **on “stupid” or insecure workstation:**
  - paper sheet of pre-computed passwords
    - password cards
  - hardware authenticator (crypto token)
- **on intelligent and secure workstation :**
  - automatically computed by an ad-hoc application
  - typical for smartphone, tablet, laptop, …



Time-based OTP

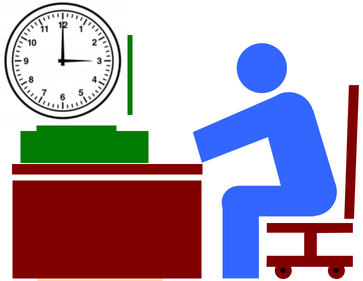
- **the password depends upon time (t) and user's secret (S_{ID}):**
  - $p(ID,t) = h(t, S_{ID})$

Verifier

***Claimant* (ID)**

authentication request →

ID, X=p(ID,t) ←



$\{ID : S_{ID}\}$

S_{ID}

$X = p\ (ID,t)$?

Time-based OTP: analysis

- **requires local computation at the subscriber**
- **requires clock synchronization (or keeping track of time-shift for each subscriber)**
- **requires time-slot and authentication window**
  - X == p(ID,t) || X == p(ID,t-1) || X == p(ID, t+1)
- **only one authentication run per time-slot**
  - typically 30s or 60s (not good for some services)
- **time attacks against subscriber and verifier**
  - fake NTP server or mobile network femtocell
- **sensitive database at the verifier**
  - see the attack against RSA SecurID

A TOTP example: RSA SecurID

- **the user (claimant) sends to the verifier (server) in clear**

  user, PIN, *token-code* (seed, time)

  or (if an authenticator with pinpad is used)

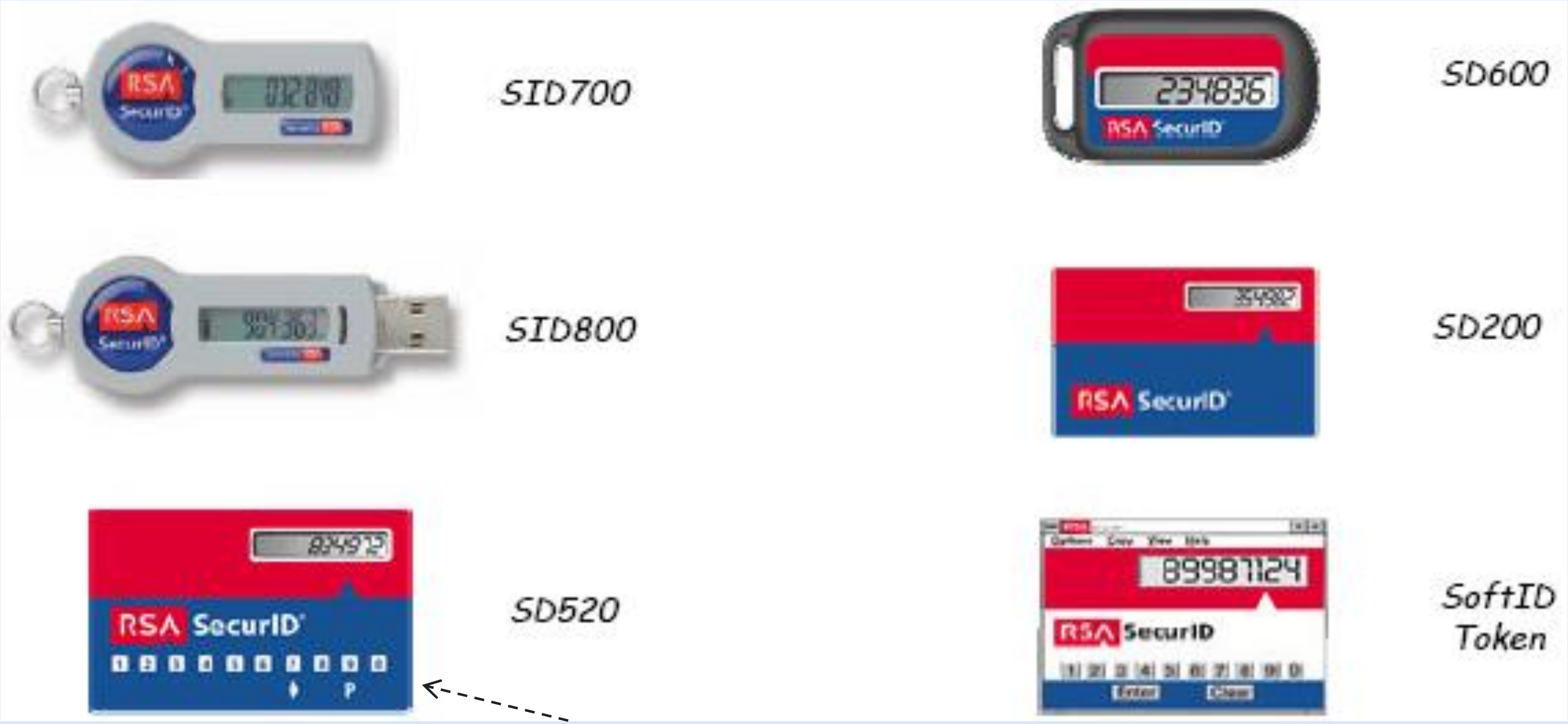
  user, *token-code** (seed, time, PIN)

- **based on *user* and PIN the server verifies against three possible token-codes:**

  $\mathbf{TC_{-1}, TC_0, TC_{+1}}$

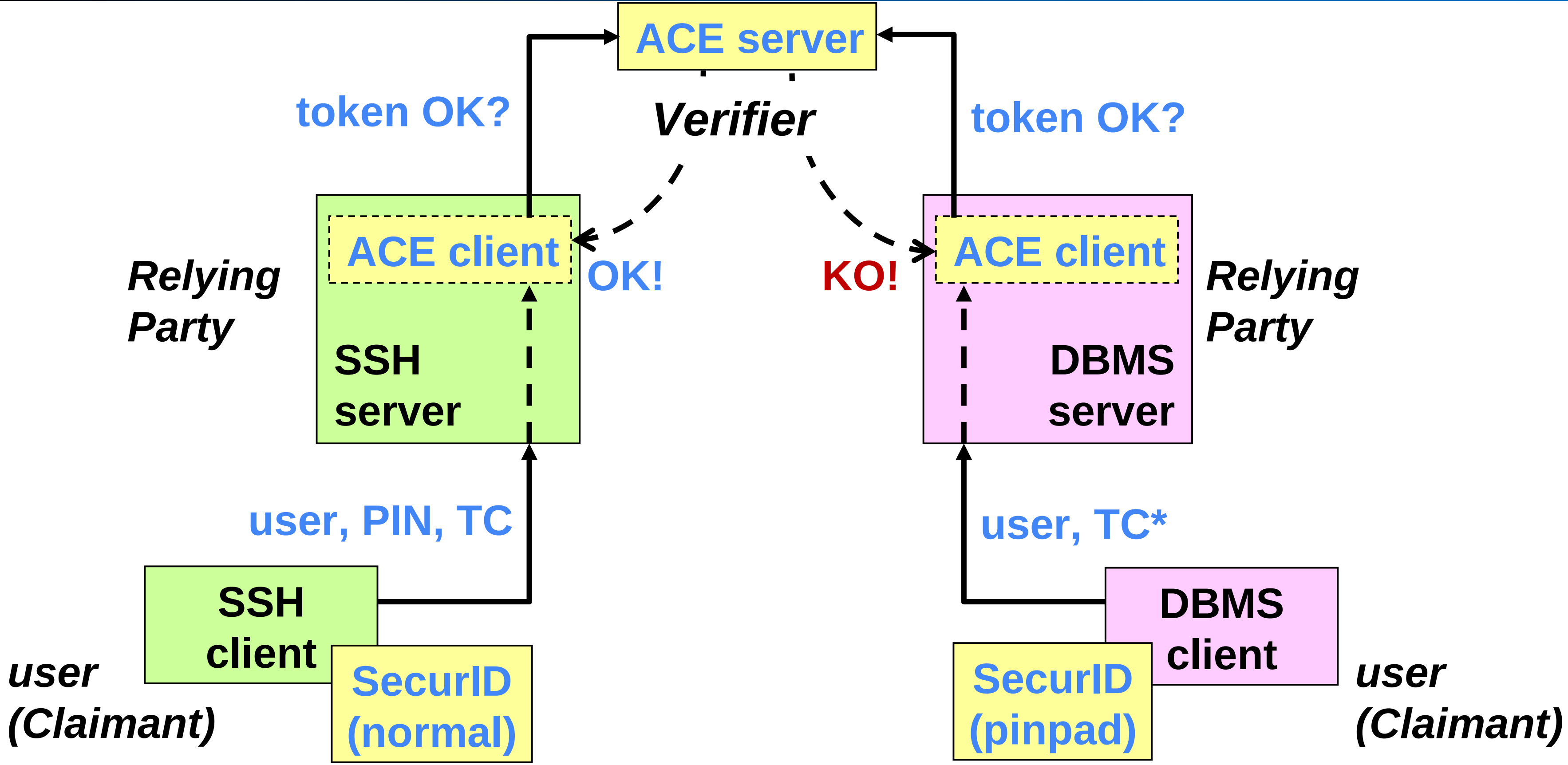
- ***duress code*: PIN to generate an alarm (to be used in case of user is forced to perform an authentication = is under threat)**
- **ACE (Access Control Engine) composed of**
  - an ACE client (installed at the Relying Party)
  - ACE server (implements the verifier)

RSA SecurID: some (recent) products



PINPAD

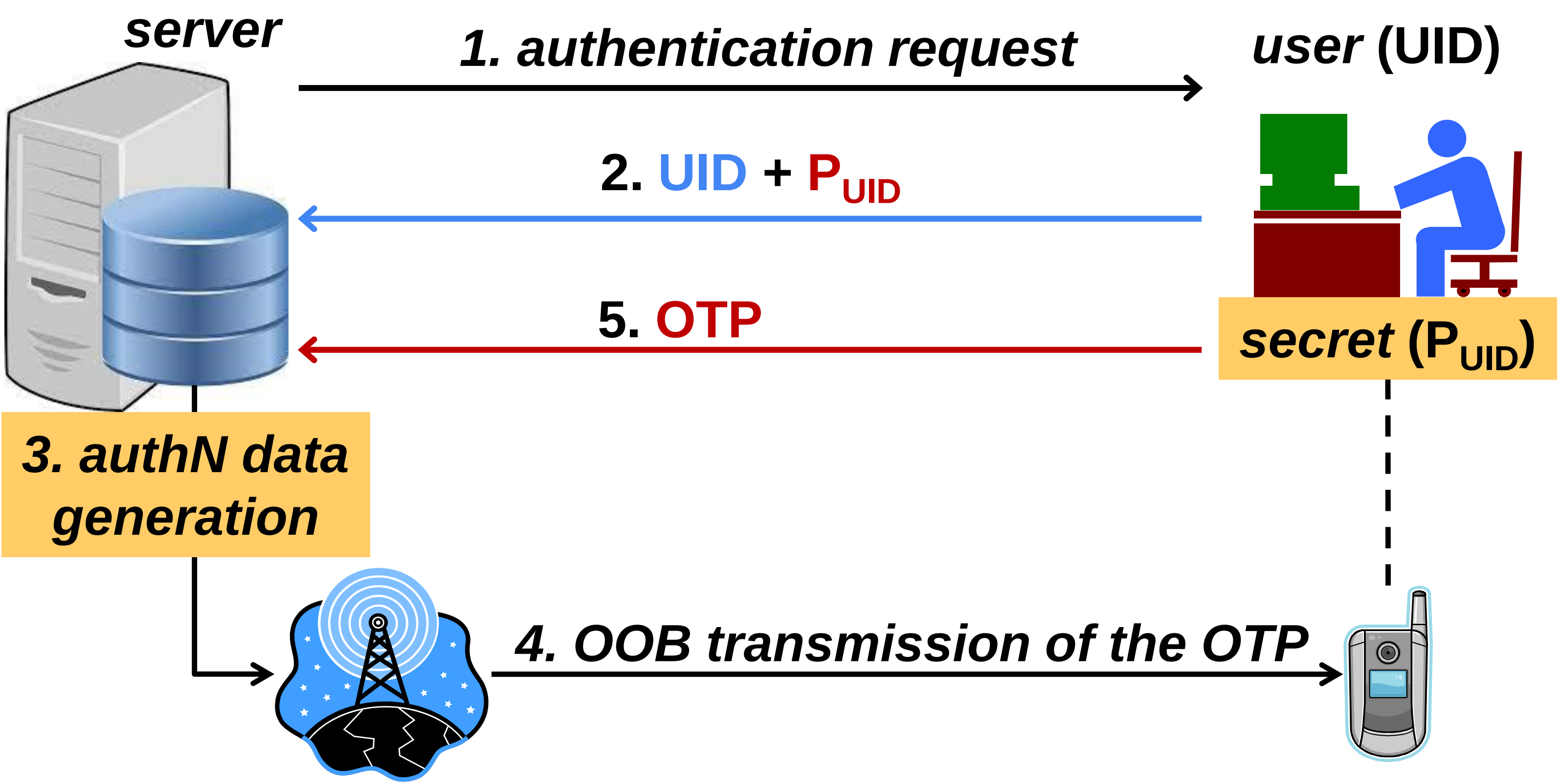
SecurID: architecture



Event-based OTP

- **uses a monotonic integer counter C as input besides the seed**
  - $p(ID,C) = h(C,S_{ID})$
- **requires local computation at the subscriber**
- **counter incremented at the subscriber (e.g. button)**
- **frequent authentication runs are possible**
- **OTP pre-computation (also by adversary who temporarily have access to the authenticator)**
- **Verifier must accommodate desynchronization**
  - the subscriber pushed button unwillingly
  - X==p(ID,C) || X=p(ID,C+1) || X==p(ID,C+2) || … ?

Out-of-band OTP



Out-of-Band OTP

- **at step 5 channel w/ server authentication needed to avoid MITM attack**
- **OOB channel frequently is text/SMS message**
  - can be attacked due to problems of VoIP, mobile user identification, and SS7 protocol
- **NIST SP800-63.B**
  - use of PSTN (SMS or voice) as OOB channel is deprecated
  - suggest using Push mechanism over TLS channel to registered subscriber device

Two-/Multi-Factors AuthN (2FA/MFA)

- **use more than one factor**
  - to increase authN strength
  - to protect authenticator
- **PIN used for authenticator protection**
  - PIN transmitted along with OTP
  - PIN entered to compute the OTP itself
  - PIN (or inherence factor) used to unlock the authenticator, very risky if:
    - lock mechanism weak
    - no protection from multiple unlock attempts
    - unlocking valid for a time window

Authentication of human beings

- **how can we be sure of interacting with a human being rather than with a program (e.g. sensing a password stored in a file)?**
- **two solutions:**
  - CAPTCHA techniques (Completely Automated Public Turing test to tell Computers and Humans Apart)
    - e.g. picture with images of distorted characters
  - biometric techniques
    - e.g. fingerprint, voice imprint

Biometric systems

- **measure of one biologic characteristics of the user**
  - physical (P)
  - behavioural (B)
  - mixed (M)
- **main characteristics being used:**
  - fingerprint (type P)
    - common on laptops and smartphones
  - face recognition (type P)
    - used by some smartphones
  - iris recognition (type P)
    - part of the eye that a contact lens cover

Biometric systems (II)

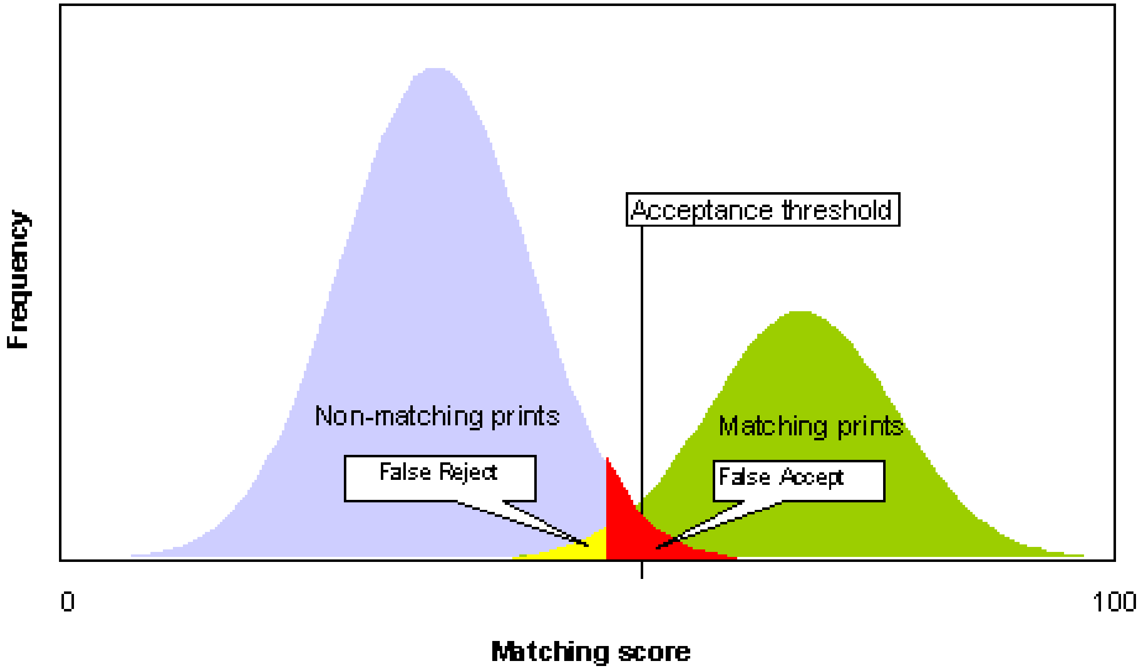
- **main characteristics being used (cont.):**
  - hand geometry (type P)
    - hand length and size, also shape of fingers and palm
  - retinal scan (type P)
    - based on patterns of retinal blood vessels
  - voice authentication (type M)
    - physical-behavioural mix
  - gait (type B)
    - characteristics related to walking
  - typing rhythm (type B)
    - keystroke patterns and timing
  - mouse patterns (type B)
    - scrolling, swipe patterns on touchscreen devices

Problems of biometric systems

- **two types of errors occur in biometric systems:**
  - false accept
    - an imposter's sample (characteristic measure) is wrongly declared to match the legitimate user's template
    - FAR = False Acceptance Rate
  - false reject
    - a legitimate user's new sample is declared to not match their own template variable biological characteristics, e.g. finger wound, voice altered due to emotion, retinal blood pattern altered due to alcohol or drug
    - FRR = False Rejection Rate
- **FAR and FRR may be partly tuned but they heavily depend on device limitations (inaccuracies) and an acceptance threshold**

FAR / FRR

- **acceptance threshold tuned: high-security vs. low-security applications**

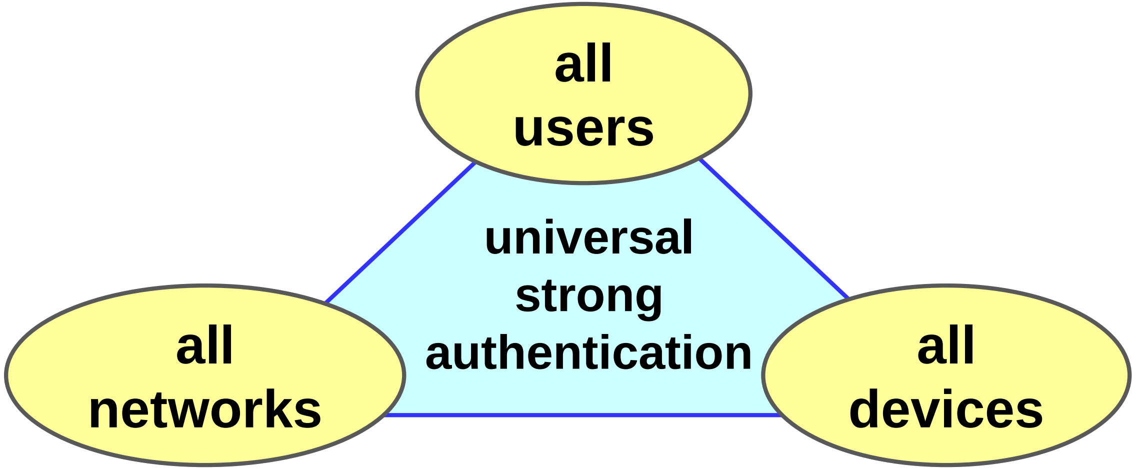


Problems of biometric systems

- **psychological acceptance:**
  - “Big Brother” syndrome (=personal data collection)
  - some technologies are intrusive and could harm
- **privacy**
  - it’s an identification
- **cannot be changed if copied**
  - hence only useful to *locally* replace a PIN or a password
- **lack of a standard API / SPI:**
  - high development costs
  - heavy dependence on single/few vendors

Authentication interoperability

- OATH (www.openauthentication.org)
- interoperability of authentication systems based on OTP, symmetric or asymmetric challenge
- development of standards for the client-server protocol and the data format on the client



OATH specifications

- **http://www.openauthentication.org/specifications**
- **HOTP (HMAC OTP, RFC-4226)**
- **TOTP (Time-based OTP, RFC-6238)**
- **OATH challenge-response protocol (OCRA, RFC-6287)**
- **Portable Symmetric Key Container (PSKC, RFC-6030)**
  - XML-based key container for transporting symmetric keys and key-related meta-data
- **Dynamic Symmetric Key Provisioning Protocol (DSKPP, RFC-6063)**
  - client-server protocol for provisioning symmetric keys to a crypto-engine by a key-provisioning server

HOTP

- K : shared secret key
- C : counter (monotonic positive integer number)
- h : cryptographic hash function (default: SHA1)
- sel : function to select 4 bytes out of a byte string
- HOTP(K,C) = sel(HMAC-h(K,C)) & 0x7FFFFFFF
- note: the mask 0x7FFFFFFF is used to set MSB=0 (to avoid problems if the result is interpreted as a signed integer)
- to generate a N digits (6-8) access code:

$$\text{HOTP-code} = \text{HOTP}(K,C) \bmod 10^N$$

TOTP

- **as HOTP but the counter C is the number of intervals TS elapsed since a fixed originT0**

  C = (T – T0) / TS

- **default (RFC-6238):**
  - T0 = Unix epoch (1/1/1970)
  - T = unixtime(now)
    seconds elapsed since the Unixepoch
  - TS = 30 seconds
  - … equivalent to C = floor (unixtime(now) / 30)
  - h = SHA1 (but may use SHA-256 or SHA-512)
  - N = 6

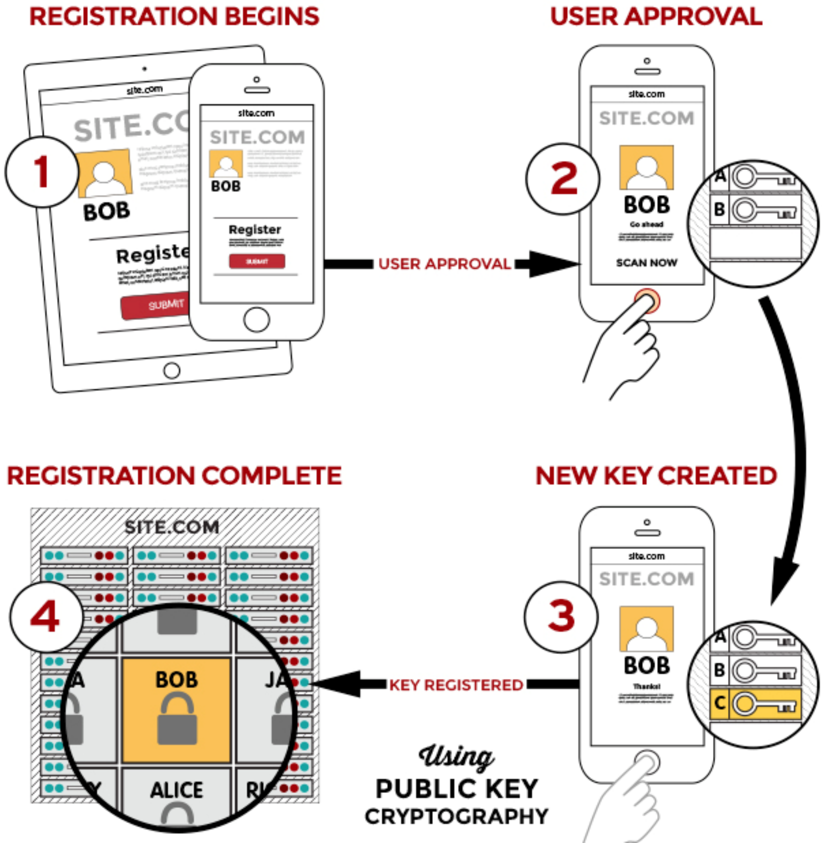
Google authenticator

- **supports HOTP and TOTP with the following assumptions:**
  - K is provided base-32 encoded
  - C is provided as uint_64
  - sel(X)
    - offset = 4 least-significant-bits of X
    - return X[offset … offset+3]
  - TS = 30 seconds
  - N = 6
  - if the generated code contains less than 6 digits then it's left padded with zeroes (e.g. 123 > 000123)

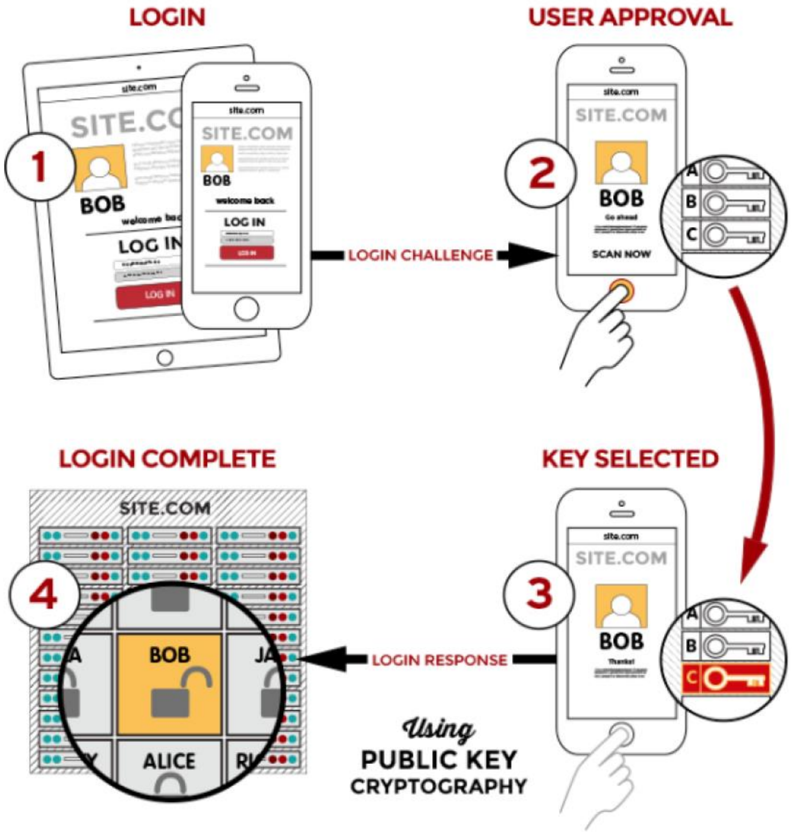
FIDO

- **Fast IDEntity Online**
- **industry standard of the FIDO Alliance for:**
  - biometric authN = passwordless user experience
  - 2-factor authN = 2nd factor user experience
- **based on personal devices capable of asymmetric cryptography**
  - for responding to an asymmetric challenge
  - for digital signature of texts
- **UAF = Universal Authentication Framework**
- **U2F = Universal 2nd Factor**
- **ASM = Authenticator-Specific Module**
- **available for major services (Google, Dropbox, GitHub, Twitter, …) and also for the cloud (AWS, Azure, …)**

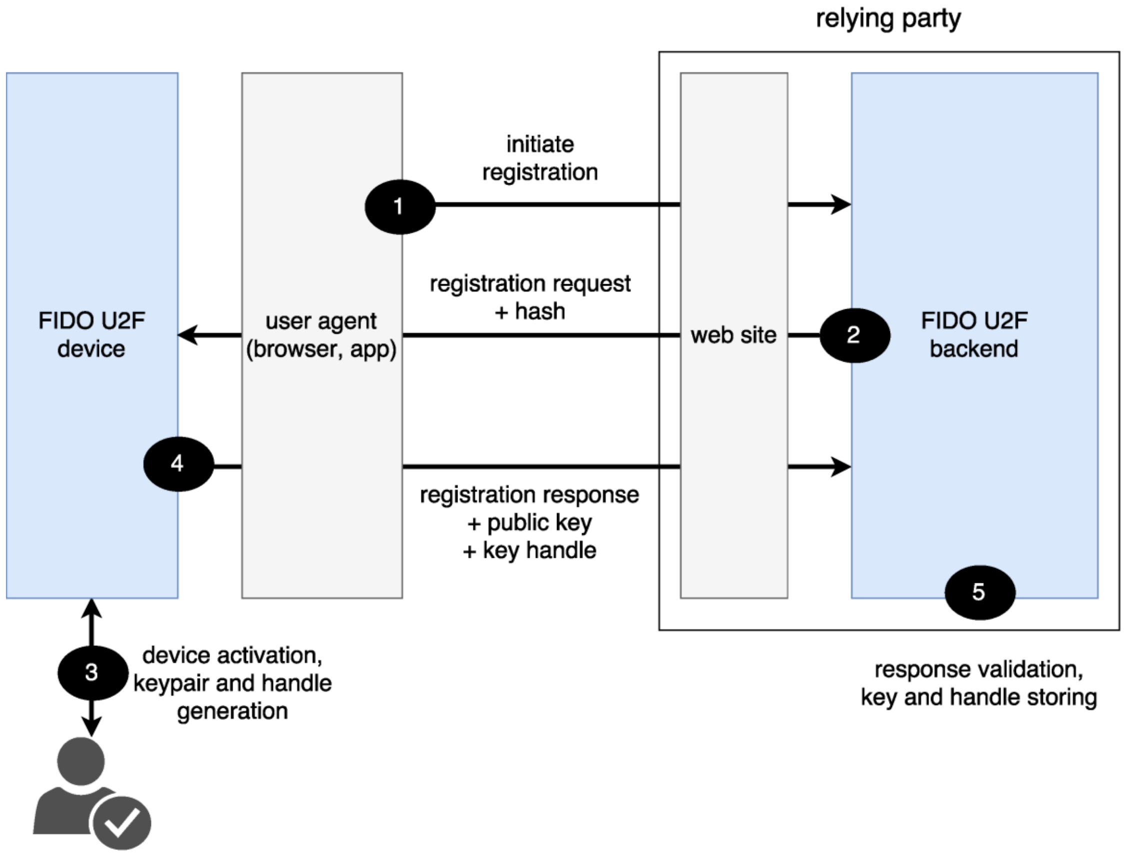
FIDO registration



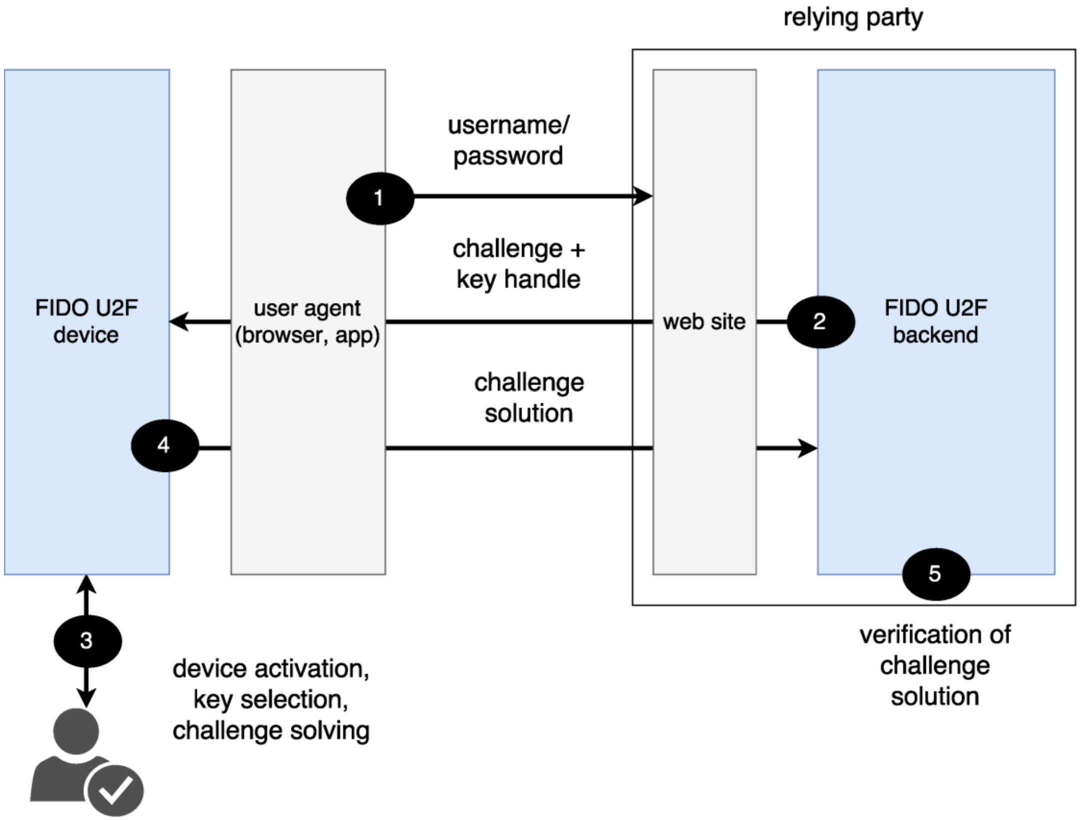
FIDO Login


LOGIN
USER APPROVAL
LOGIN CHALLENGE
KEY SELECTED
LOGIN COMPLETE
LOGIN RESPONSE
1
2
3
4
SITE.COM
BOB
LOG IN
SCAN NOW
A
B
C
ALICE
Using
PUBLIC KEY
CRYPTOGRAPHY

FIDO U2F registration



FIDO U2F authentication



FIDO: other characteristics

- **biometric techniques**
  - local authentication method to enable the FIDO keys stored on the user device
- **secure transactions**
  - digital signature of a transaction text (in addition to the response to the challenge)
- **FIDO backend (or server)**
  - to enable the use of FIDO on an application server
- **FIDO client**
  - to create and manage credentials FIDO on a user device

FIDO: security and privacy

- **security**
  - strong authentication (asymmetric cryptography)
  - no 3rd party in the protocol
  - no secrets on the server side
  - biometric data (if used) never leaves user device
  - no phishing because authN response can’t be reused:
  - it’s a signature over various data, including the RP identity

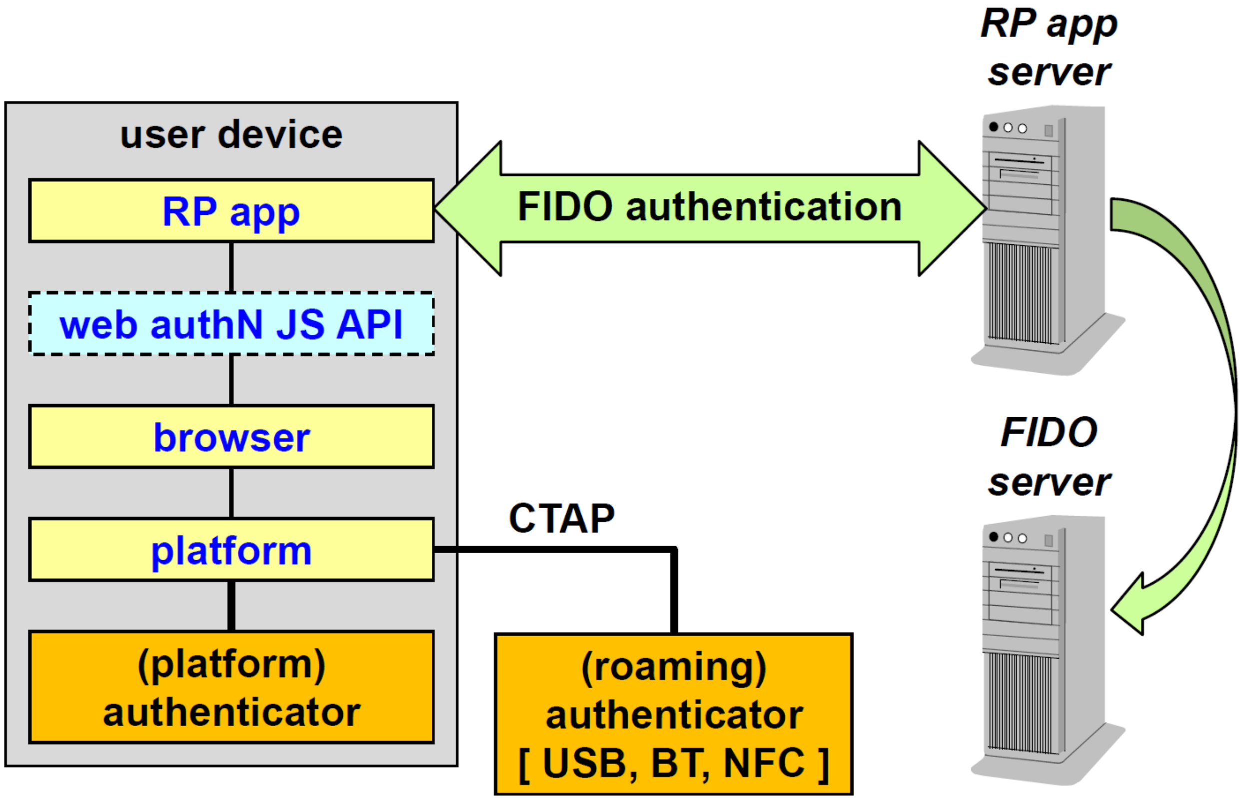
FIDO: security and privacy - II

- **privacy**
  - since one new key-pair is generated at every registration, we obtain no link-ability among:
    - different services used by the same user
    - different accounts owned by the same user
  - there is no limit because private keys are not stored in the authenticator but recomputed as needed based on an internal secret and RP identity

Fido: evolution

- Feb.2013: FIDO alliance launched
- Dec.2014: FIDO v1.0
- Jun.2015: Bluetooth and NFC as transport for U2F
- Nov.2015: submission to W3C of the Web API for accessing FIDO credentials
- Feb.2016: W3C creates the Web Authentication WG to define a client-side API that provides strong authentication functionality to Web Applications, based on the FIDO Web API

FIDO 2.0


user device
RP app
web authN JS API
browser
platform
(platform)
authenticator
FIDO authentication
CTAP
(roaming)
authenticator
[USB, BT, NFC]
RP app
server
FIDO
server