Authenticatie

Cyberboswachters

Tim Dams

Authenticatie vs Autorisatie

Concept	Vraag
Authenticatie	*Wie ben je?* — Bewijs van identiteit
Autorisatie	*Wat mag je?* — Toekennen van rechten

> **Tip**
>
> Dit hoofdstuk: focus op **authenticatie** — hoe gaan we als beheerders veilig om met login-informatie?

De geheime sleutel als authenticatie

- **Weten van de geheime sleutel** = eerste vorm van authenticatie
- Login = combinatie **gebruikersnaam + wachtwoord**
- Wachtwoord = in essentie je **geheime sleutel**

> **Belangrijk**
>
> Als beheerder is het **essentieel** om uitermate veilig om te gaan met de wachtwoorden van gebruikers!

Veilige wachtwoorden: de regels

Regels die continu **niét** gehanteerd worden:

- **Hergebruik nooit** een wachtwoord — één per service
- Gebruik geen wachtwoorden uit **dictionaries** — overweeg random gegenereerde wachtwoorden
- Minimum **16 tekens** lang
- Combinatie van **cijfers, letters** (groot + klein) **en leestekens**

Hoe stropers aan wachtwoorden komen

Niet altijd via de database! Denk aan de McCumber kubus: **technologie is maar één aspect**.

Aanval	Werkwijze
Password spraying	Veelgebruikte wachtwoorden testen op veel accounts (*"spray and pray"*)
(Spear) phishing	Nep-mail/bericht → malware of fake login. Bij spear phishing: gericht op één doelwit
Keyloggers	Alle toetsaanslagen opnemen en later analyseren

> ⚠ **Waarschuwing**
>
> Spear phishing is heden ten dage één van **dé** social engineering aanvallen bij uitstek.

Online vs. offline aanvallen

Online aanvallen

- Actief inloggen of wachtwoord bij de bron onderscheppen
- Password spraying, phishing, keyloggers

Offline aanvallen

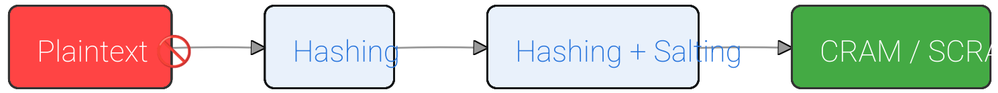
- Aanvaller heeft (read-only) toegang tot de **wachtwoord-databank**
- Via SQL injection, insider, datalek, ...

Belangrijk

Ga er van uit dat het gebeurt! De komende slides tonen hoe we als beheerder de schade bij zo'n lek tot een minimum beperken.

Hoe wachtwoorden opslaan?

We gaan **bottom-up**: van naïef naar veilig.



Stap 1: Paswoorden als plaintext

- Database met twee kolommen: gebruikersnaam + wachtwoord
- Wachtwoord staat er **zoals het is**
- Aanvaller steelt database → **alle wachtwoorden** in handen!

User	Password
alice	`welcome123`
bob	`qwerty`
carol	`password1`

Belangrijk

Paswoorden mogen NOOIT als plaintext in een databank staan!

Hoe herken je plaintext-opslag?

- Gebruik de *“Ik ben m’n wachtwoord vergeten”*-knop
- Krijg je een e-mail met **jouw originele wachtwoord**? → plaintext opslag!

> ⚠ **Waarschuwing**
>
> Een service zou **NOOIT** jouw wachtwoord moeten kunnen zien. Uiteindelijk mag **je wachtwoord nooit je computer verlaten**.

Wie zou nu zo dom zijn?!

Facebook, 2019: wachtwoorden van honderden miljoenen gebruikers in **plaintext** bewaard

- **200 – 600 miljoen** gebruikers getroffen
- ~**20.000** Facebook-medewerkers hadden toegang
- Jarenlang onopgemerkt gebleven

> ⚠ **Waarschuwing**
>
> Zelfs de grootste techbedrijven vallen in deze klassieke val. Bron: The Verge (2019)

Stap 2: Paswoord hashing

- Gebruiker genereert **hash** van wachtwoord op lokaal systeem
- **Hash** wordt verstuurd (niet het wachtwoord zelf)
- Server vergelijkt ontvangen hash met hash in database

Maar: vatbaar voor **pass-the-hash** aanval!

- Aanvaller hoeft enkel een geldige **gebruikersnaam + hash** te capteren
- Geen kennis van het originele wachtwoord nodig

Probleem met hashing

Tip

Door een *secure hash* te genereren creëren we tekst die **niet omkeerbaar** is. Maar:

1. **Collisions**: twee verschillende wachtwoorden → dezelfde hash
2. **Hergebruik**: twee gebruikers met hetzelfde wachtwoord → dezelfde hash

Oplossing: **salting** (zie verder)

Waarschuwing

Veel websites genereren de hash **aan serverzijde**, maar dan wel over een beveiligde **TLS tunnel**.

Rainbow table attack: het dilemma

Als een database gestolen wordt, moet de aanvaller hashes kraken:

Optie 1: Ter plekke hashen

- Algoritmes zoals *bcrypt*, *scrypt* en *pbkdf2* zijn **by design traag**
- → **Timebottleneck**

Optie 2: Vooraf precomputen

- Alle mogelijke hashes op voorhand berekenen
- Lijst wordt **gigantisch** groot
- → **Memorybottleneck**

> **Opmerking**
>
> Een **rainbow table** biedt een compromis tussen beide bottlenecks.

Even concreet: 8 kleine letters

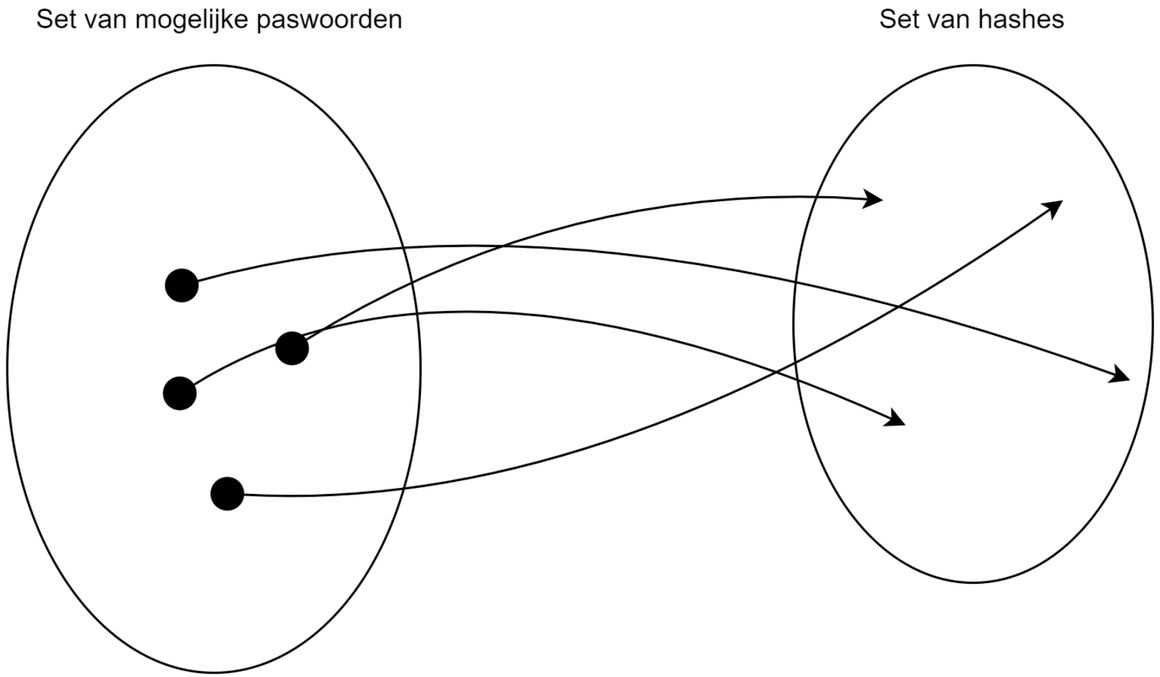
Aspect	Waarde
Mogelijke wachtwoorden	$26^8 \approx 2 \cdot 10^{11}$
Bruteforce @ 1 miljoen/s	~2,4 dagen
Bruteforce @ 2,8 miljard/s (GPU, 2011)	**minuten**
Opslag alle hashes	~1,46 TB
Alle Google-servers samen	$\sim 10^{15}$ bytes (1 PB)

Opmerking

Puur bruteforcen *of* puur precomputen loont nauwelijks. Rainbow tables zoeken bewust een **compromis** tussen beide uitersten.

Rainbow table: het concept

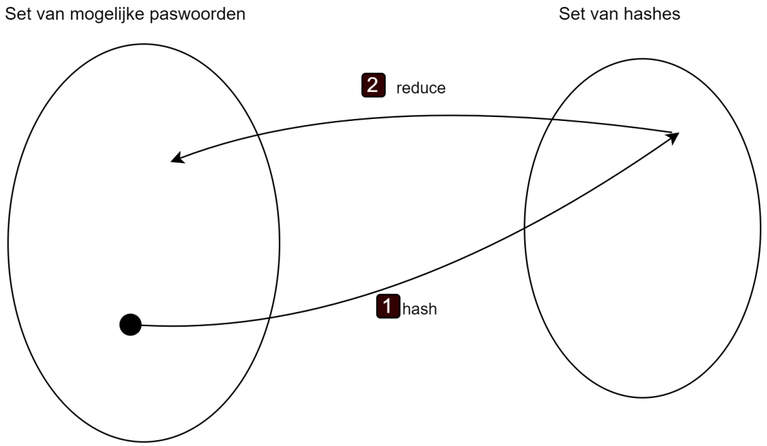
- Een **gecomprimeerde** tabel van precomputed hashes
- Niet alle hashes bewaren, maar toch een grote set bestrijken
- Vergelijk: getallen 1 tot 101 bewaren als *“start=1, eind=101, stap=2”*



Ieder wachtwoord mapt naar exact één hash.

Rainbow table: opbouwen

1. Kies een **startpunt** (wachtwoord uit de set)
2. Genereer de **hash**
3. Pas een **reduction functie** toe → terug naar een wachtwoord

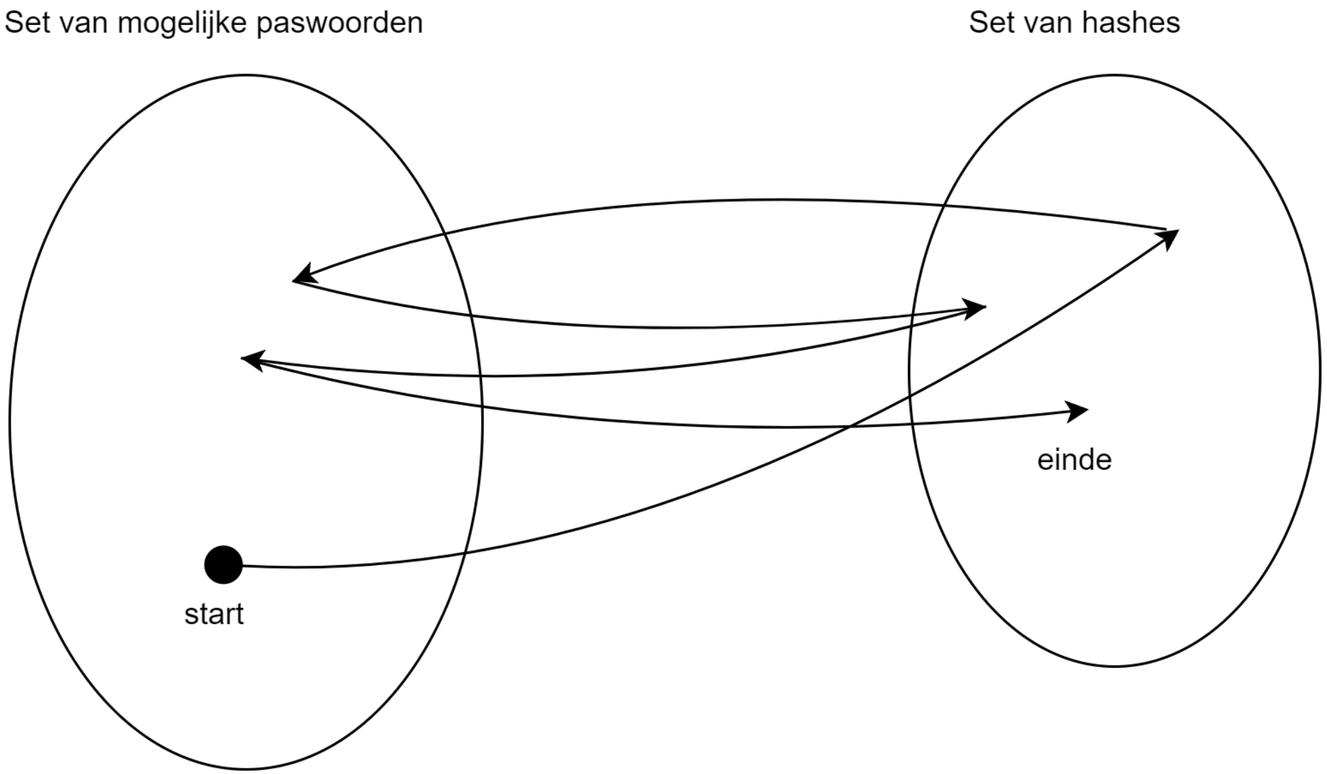


Van de hash via de reductiefunctie terug naar een ander wachtwoord.

> **Tip**
>
> Reductiefunctie: bv. som van ASCII-waarden → index in de wachtwoordenset.

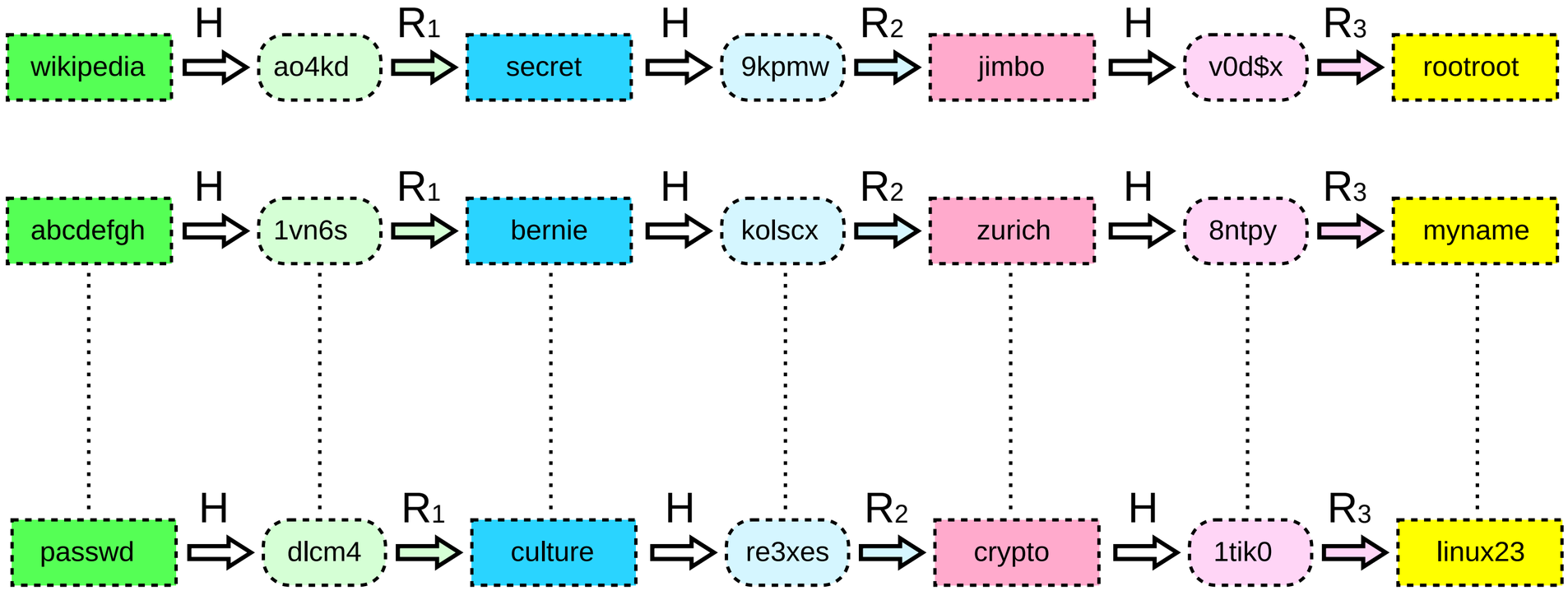
Rainbow table: de chain

- Herhaal hash + reductie **duizenden keren** → een *chain*
- Bewaar enkel het **startpunt** en de **laatste hash**



Voorbeeld van een chain.

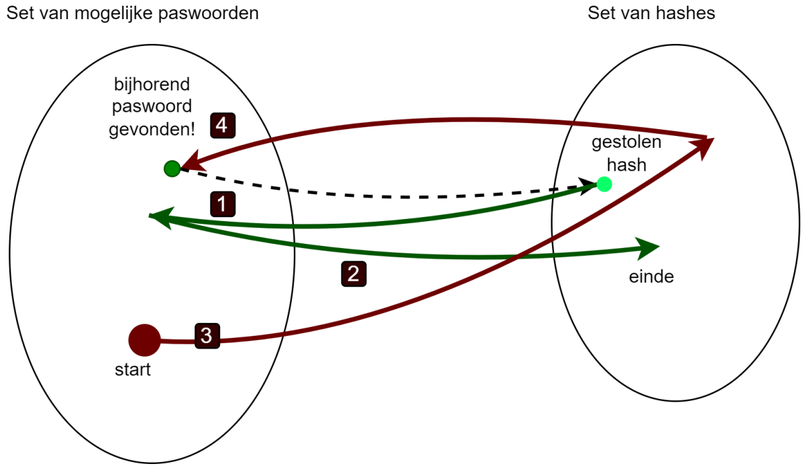
Meerdere chains met verschillende reducties



Drie parallelle chains, elk met een eigen reductiefunctie (kleur). Bron: Wikimedia Commons, CC BY-SA 2.5.

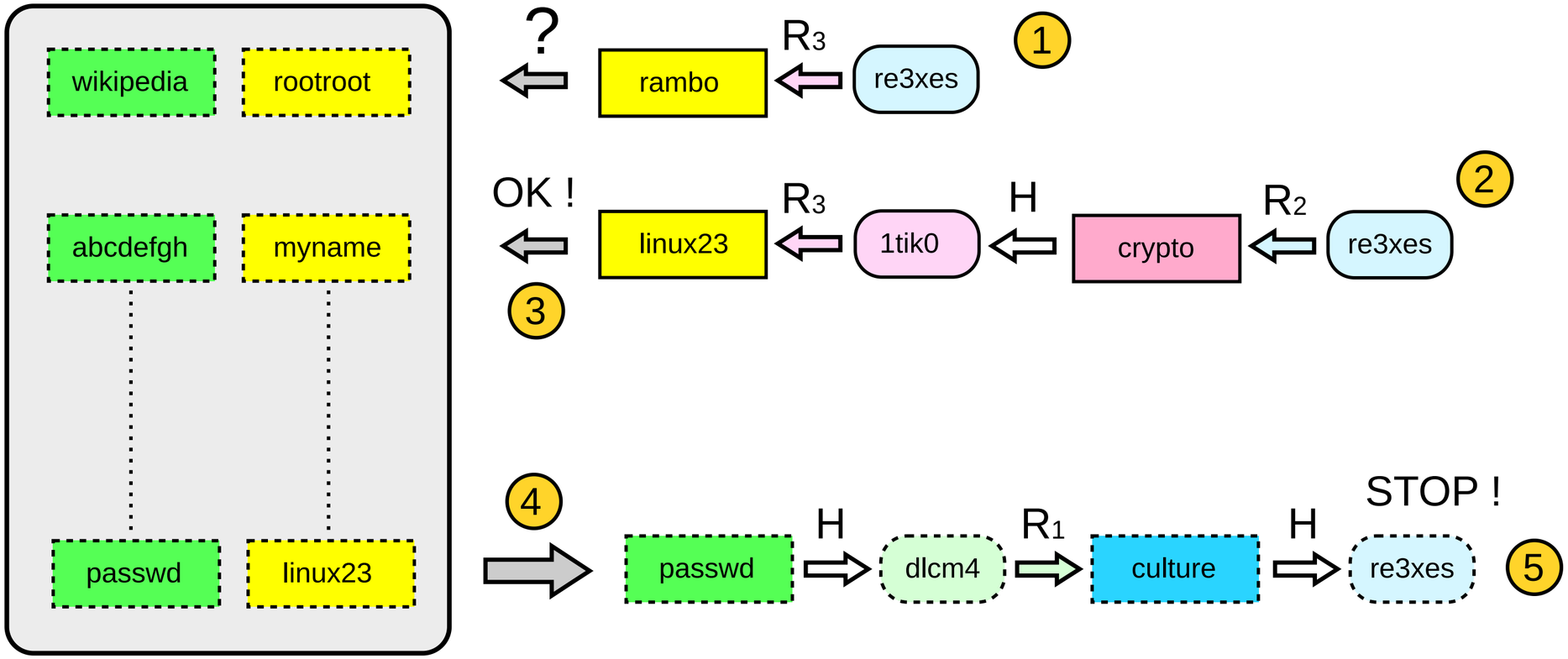
Rainbow table: kraken

1. Start met de **gestolen hash**
2. Pas reductie + hash toe tot je een **bekend eindpunt** vindt
3. Neem het **startpunt** van die chain
4. Herhaal hash + reductie tot je de gestolen hash bereikt
5. Één stap terug = het **wachtwoord**!



Het kraken van een hash met een rainbow table.

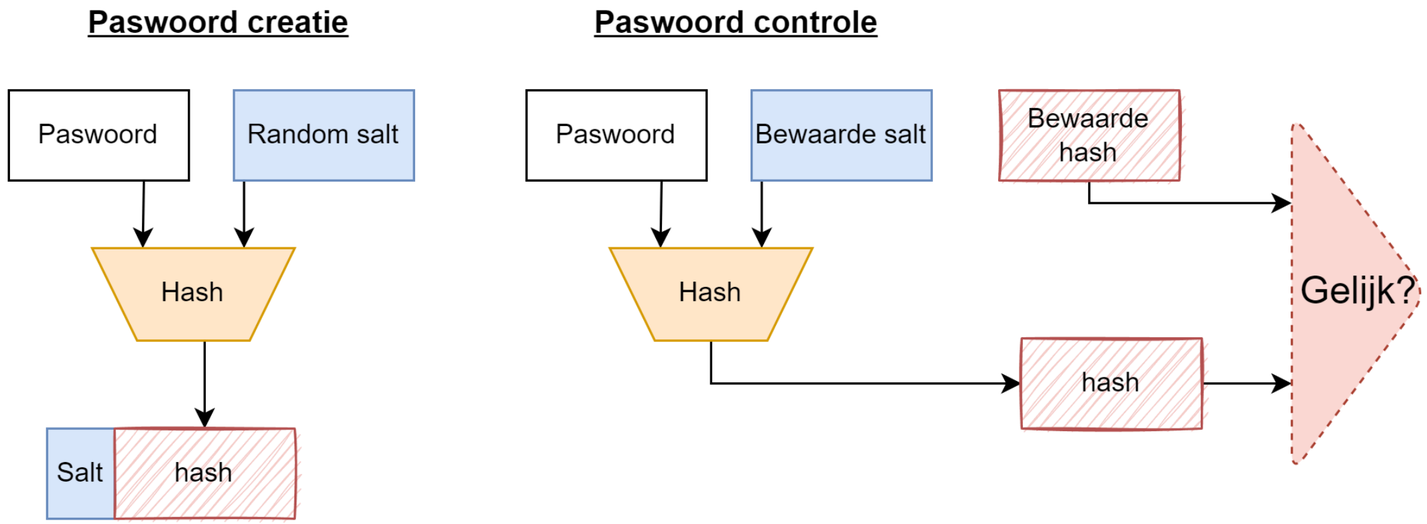
Het volledige zoekproces in één beeld



Van gestolen hash `re3xes` → eindpunt `linux23` vinden → chain herberekenen vanuit `passwd` → wachtwoord `culture` ligt één stap vóór de gestolen hash. Bron: Wikimedia Commons, CC BY-SA 4.0.

Stap 3: Salting

- **Salt** = random extra stuk dat je toevoegt aan het wachtwoord **vóór** het hashen
- Unieke salt per gebruiker → zelfs identieke wachtwoorden geven **andere hashes**
- Rainbow tables worden **onbruikbaar** (set wordt te groot)



Het salting proces.

Salting: de database

Veld	Beschrijving
Gebruikersnaam	Identificatie
Salt	Random waarde (min. 32 bits)
Hash	Hash van wachtwoord + salt

> ⚠ **Waarschuwing**
>
> Merk op: ook nu zijn we nog steeds **niet beschermd** tegen pass-the-hash aanvallen!

Mimikatz

- Origineel: demo dat Microsoft authenticatie **onveilig** was
- Snel overgenomen door digitale **stropers**
- Vindt en hergebruikt *authentication tickets* op Windows

Aanval	**Wat?**
Pass-the-hash	Windows NTLM hashes
Pass-the-ticket	Kerberos tickets
Golden Ticket	KRBTGT account (vervalt niet!)
Pass-the-cache	Mac, Unix, Linux tickets

Mimikatz en NotPetya

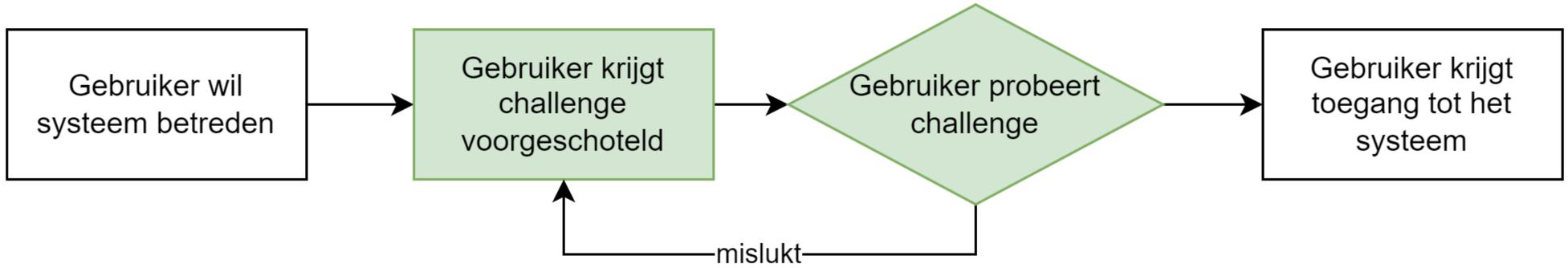
- Mimikatz is **automatiseerbaar** → gevaarlijk op schaal
- **NotPetya** (2017): ransomware-aanval op Oekraïne
  - → Aangepaste Mimikatz-variant om zichzelf te **verspreiden** over het netwerk
  - → Kon inloggen op andere systemen via gestolen hashes en tickets

> **Tip**
>
> Petya en NotPetya werden getraceerd naar **Sandworm**, een Russische hackinggroep onder de GRU (militaire inlichtingendienst).

CRAM

Challenge-Response Authentication Mechanism

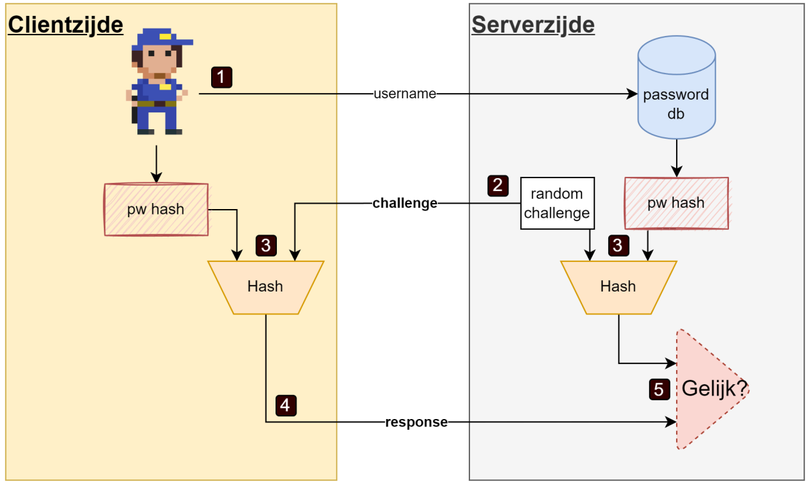
- Gebruiker krijgt een **challenge** (vraag) en moet een geldig **response** (antwoord) geven
- Wachtwoord-login = een vorm van CRAM
- Andere voorbeelden: CAPTCHA's, irisscan, …



CRAM.

CRAM: de flow

1. Gebruiker stuurt **username** → “ik wil inloggen”
2. Server genereert **random challenge** → stuurt terug
3. Beiden berekenen: **hash(challenge + password hash)**
4. Client stuurt zijn **response** naar server
5. Server vergelijkt → **match = toegang**



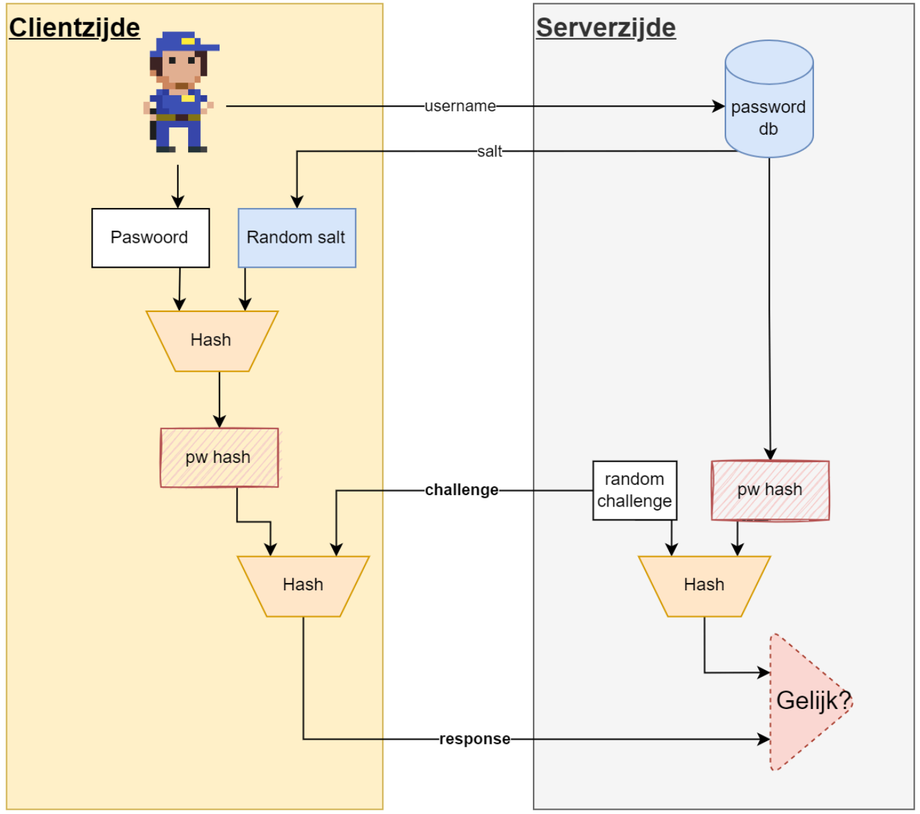
CRAM flow.

SCRAM: Salted Challenge-Response Authentication Mechanism

Lost **pass-the-hash** op! Twee garanties:

1. De salted hash wordt **NOOIT** verzonden
2. **Geen replay** aanval mogelijk

Server stuurt ook de **bewaarde salt** naar de client → client moet zelf de hash berekenen.

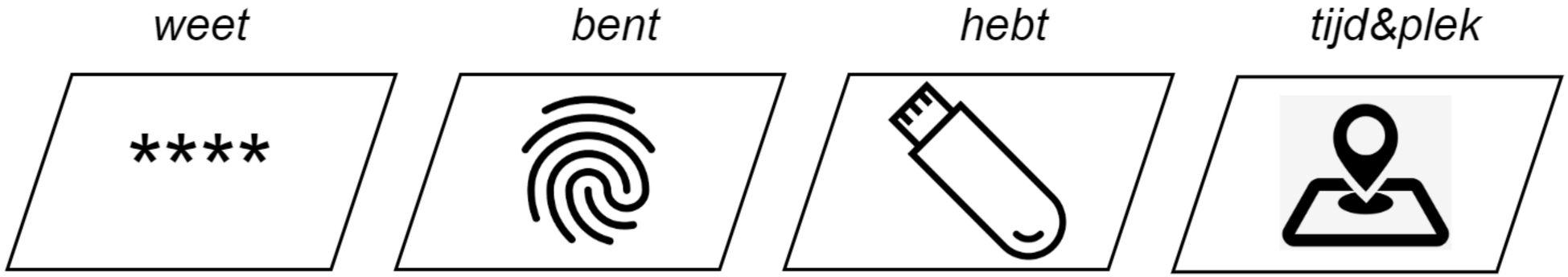


SCRAM flow.

Multifactor authentication (MFA)

Vier **factoren** om identiteit te controleren:

Factor	Voorbeeld
lets wat je **weet**	Wachtwoord, pincode, rijksregisternummer
lets wat je **bent**	Vingerafdruk, irisscan (*biometrics*)
lets wat je **hebt**	Smartphone, USB-key
Waar/wanneer je bent	IP-adres, tijdstip van login



MFA in de praktijk

- **2FA** = minstens twee factoren vereist
- Meer factoren = **veiliger**, maar ook **minder gebruiksvriendelijk**
- Systemen passen MFA **dynamisch** aan:

> **Opmerking**
>
> Je woont in België maar iemand logt in vanuit de Azoren met jouw wachtwoord? → Google vraagt **extra verificatie** (andere factor).

Factor 1: lets wat je weet (paswoorden)

Het grote probleem met *dingen weten*:

- Je kan ze **vergeten** → niet meer inloggen
- Anderen kunnen ze **te weten komen** → identiteitsdiefstal

> ⚠ **Waarschuwing**
>
> Technologisch het **eenvoudigst** te implementeren, maar de **minst veilige** factor vanuit social engineering standpunt.

Factor 2: Iets wat je bent (biometrics)

“We zijn allemaal uniek”

Veelgebruikte biometrieken:

- **Vingerafdruk**
- **Iris**scan
- **Stem**
- **Gezicht** (3D stereo camera)
- Maar ook: typsnelheid, manier van wandelen (*gait*), …

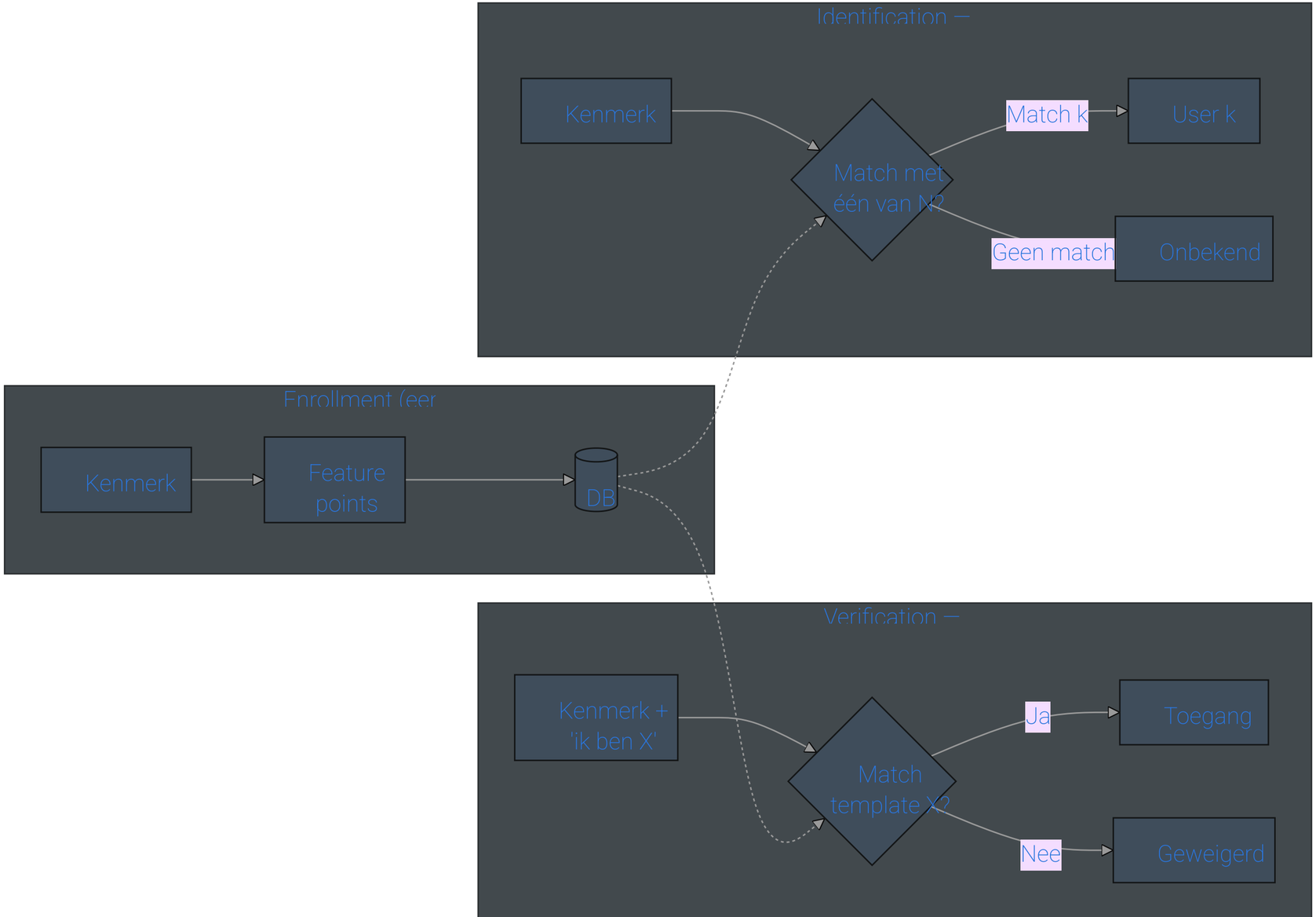
Biometrics: opslag

- **Feature points** = unieke waarden per persoon
- Opgeslagen als korte sequentie van getallen in de database
- Bij login: (quasi) dezelfde feature points → **match**

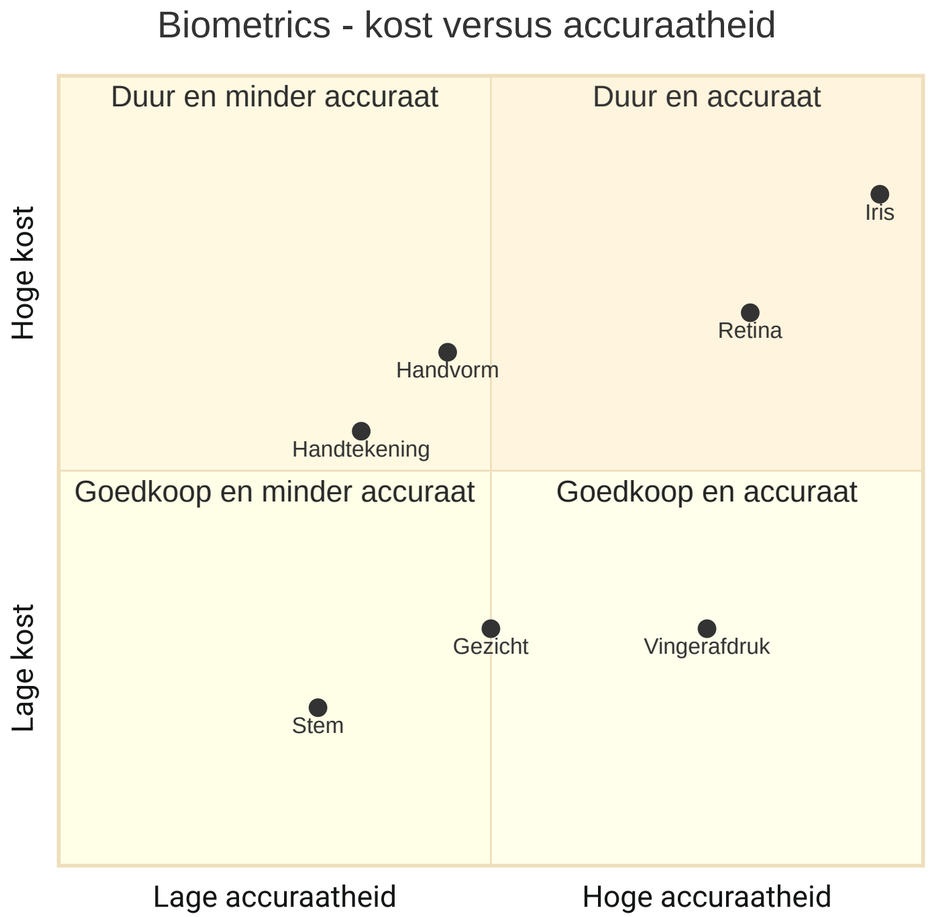
> ⚠ **Waarschuwing**
>
> **Privacy** is een heikel punt! In India werd de **Aadhaar**-database (irisscan + 10 vingerafdrukken van 1.1 miljard burgers) in 2018 gehackt.

> 💡 **Tip**
>
> Biometrics dienen niet alleen als extra **authenticatie**factor, maar ook als **identificatie**: een aanvaller kan niet zomaar een andere identiteit claimen als een vingerafdrukscanner wordt gebruikt.

Biometrics: drie fasen



Biometrics: kost vs. accuraatheid



Opmerking

Factor 3: Iets wat je hebt (hardware)

- Fysiek object = **moeilijk** na te maken
- In essentie: bevat een **veel langer wachtwoord** dan een mens kan onthouden

Twee grote families:

Type	Voorbeeld
Smartphone	Authenticator app (Google Authenticator, …)
USB-sleutel	YubiKey, Titan Key, …

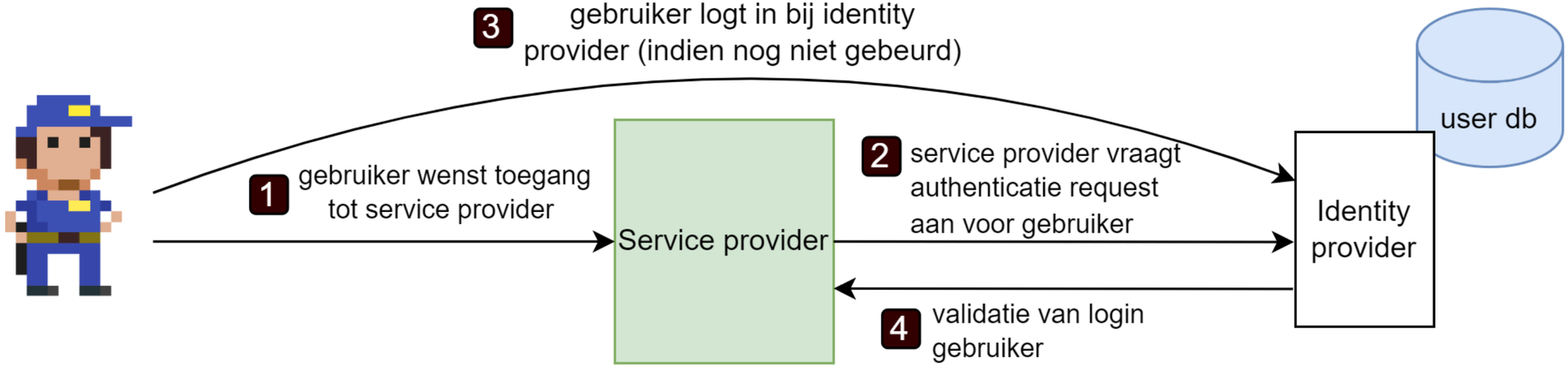
> ⚠ **Waarschuwing**
>
> Nadeel: fysiek object kan je **verliezen** of het kan **stuk gaan**.

Single Sign-On (SSO)

- **SSO** = éénmaal inloggen → toegang tot **meerdere** applicaties
- Voorbeeld: inloggen bij Google → meteen toegang tot Gmail, YouTube, Drive, …
- Authenticatie wordt uitbesteed aan een centrale **identity provider** (IdP)
- De applicatie = **service provider**, vertrouwt op de bevestiging van de IdP

Federation

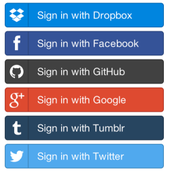
- Zelf logindata beheren? → grote **kans op fouten**
- **Federation**: SSO over de grenzen van organisaties heen
- Gebruikers loggen in via bestaand account (Google, Facebook, …)
  - → Jouw site = **service provider**
  - → Google/Facebook = **identity provider**



Een vereenvoudigd SSO proces.

Delegation vs Federation

Concept	Beschrijving
Delegation	Verplicht inloggen met **één specifieke** third-party (bv. Facebook)
Federation	Éénder welke compatibele third-party (bv. via **OpenID**)



Typische SSO knoppen.

> **Tip**
>
> **OAuth** (*open authorization*) is een gestandaardiseerde manier om aan authenticatie te doen.

> **⚠ Waarschuwing**
>
> Bij federatie: in hoeverre **vertrouw** je Google of Meta met jouw (login)data? → **privacy**!

WebAuthn en Passkeys

- **WebAuthn**: inloggen **zonder wachtwoord** via een *authenticator* (USB-sleutel, smartphone)
- **Passkeys** = combinatie van **asymmetrische crypto** + **hardware authenticatie**

Belangrijke termen:

Term	Beschrijving
FIDO Alliance	Organisatie achter de standaarden (*Fast IDentity Online*)
WebAuthn	Web Authentication API
FIDO2	Standaard die WebAuthn ondersteunt
U2F	Universal 2nd Factor (oudere standaard)

Tip

Duidelijke uitleg over passkeys: youtu.be/cFbc6yMETh4

Passkey registratie

1. Gebruiker kiest **passkey** i.p.v. wachtwoord
2. Gebruiker identificeert zich lokaal (fingerprint, PIN, YubiKey, …)
3. Toestel genereert **sleutelpaar**:
    - **Private sleutel** → blijft op het toestel
    - **Public sleutel** → naar de website

> **Belangrijk**
>
> De website heeft **nooit** toegang tot de private sleutel! *"Het wachtwoord verlaat nooit het apparaat."*

Passkey registratie: het attestation object

De publieke sleutel gaat niet *zomaar* over de draad:

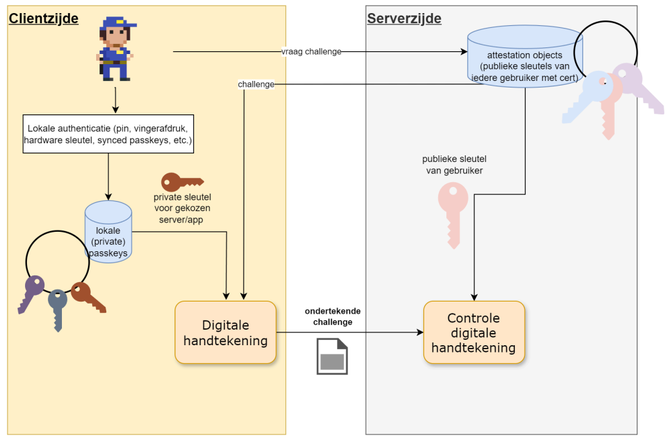
- Verpakt in een **attestation object**:
  - → Publieke sleutel
  - → **Signed challenge** (bewijs van authenticiteit)
  - → Credential ID
  - → Certificaat

> ⚠ **Waarschuwing**
>
> Private sleutels worden opgeslagen op de authenticator. Verlies je die? → **accounts kwijt!** Gebruik een **password manager** met **synced passkeys**.

Inloggen met een Passkey

Gebaseerd op **public key crypto**:

1. Server stuurt een **challenge**
2. Gebruiker encrypteert de challenge met zijn **private sleutel**
3. Server decrypteert met de bewaarde **publieke sleutel**
4. Lukt dit? → gebruiker is **geauthenticeerd**!



Het login proces met een passkey.

Authenticator apps en TOTP

- Apps zoals **Google Authenticator**, Microsoft Authenticator, Authy
- Genereren om de **30 seconden** een nieuwe **zescijferige code**
- Werken **zonder internetverbinding**!

> **Opmerking**
>
> Hoe kan een app op jouw telefoon dezelfde code genereren als de server verwacht? → **TOTP** (*Time-based One-Time Password*)

TOTP: de gedeelde geheime sleutel

Bij het koppelen van de authenticator app (**Dit is het enige moment waarop de sleutel wordt uitgewisseld. Daarna communiceren app en server nooit meer rechtstreeks**):

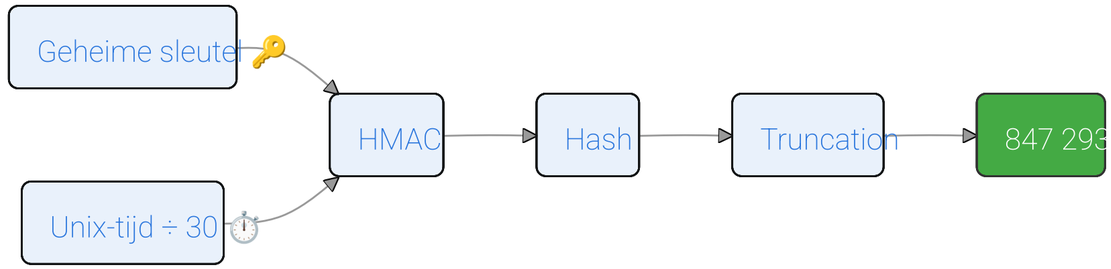
1. Website genereert een **willekeurige geheime sleutel** (typisch 160 bits)
2. Sleutel wordt getoond als **QR-code** → je scant deze met de app
3. Zowel server als app bewaren nu **dezelfde sleutel**

> ⚠ **Waarschuwing**
>
> De QR-code bevat de **volledige geheime sleutel**. Geen screenshots maken! Wie de sleutel heeft, kan jouw codes genereren.

TOTP: het algoritme

Twee ingrediënten: **gedeelde sleutel** + **huidige tijd**

1. Neem de **Unix-tijd** (seconden sinds 1 jan 1970), deel door **30**, rond af naar beneden → de **tijdstap**
2. Bereken **HMAC**(geheime sleutel, tijdstap) → lange hash
3. Extraheer via *dynamic truncation* een **zescijferig getal** → de code op je scherm



TOTP: waarom is het veilig?

Tip

Servers accepteren meestal ook de code van het **vorige en volgende** tijdsblok (~90 sec marge) om klokverschillen op te vangen.

TOTP: waarom is het veilig?

Eigenschap	Uitleg
Eenmalig	Code is slechts **30 seconden** geldig — onderschepte code is snel vervallen
Offline	Na registratie is **geen internet** meer nodig — enkel de tijd synchroniseert
Onvoorspelbaar	Zonder de geheime sleutel zijn toekomstige codes **onmogelijk** te berekenen

> ⚠ **Waarschuwing**
>
> TOTP is **niet onfeilbaar**: een real-time phishing aanval (proxy tussen jou en de echte site) kan wachtwoord + TOTP-code tegelijk onderscheppen. **Passkeys** zijn hiertegen beter beschermd (domeingebonden).

Drie gouden regels

Alles in dit hoofdstuk valt terug te brengen tot:

1. Het wachtwoord mag nooit **client → server** gestuurd worden
2. Het wachtwoord mag nooit **server → client** gestuurd worden
3. Het wachtwoord mag nooit **in leesbare vorm op de server** bewaard worden

> **Tip**
>
> **Passkeys** gaan nog een stap verder: het wachtwoord (de private sleutel) verlaat zelfs de *authenticator* niet!

Conclusie

- **Authenticatie** = bewijzen wie je bent; **autorisatie** = wat mag je
- Wachtwoorden: nooit plaintext → **hashing + salting** als minimum
- **CRAM/SCRAM** beschermen tegen pass-the-hash aanvallen
- **MFA** combineert meerdere factoren (weten, zijn, hebben, waar/wanneer)
- **TOTP** (authenticator apps): gedeelde sleutel + tijd → eenmalige codes
- **Passkeys** (WebAuthn) = de toekomst: asymmetrische crypto zonder wachtwoorden

> **Belangrijk**
>
> Hoe meer factoren, hoe veiliger. Maar de **gebruiksvriendelijkheid** blijft altijd een afweging!