

Authenticatie

Cyberboswachters

Tim Dams

Deze cursus werd gemaakt met hulp van AI-assistentie. Sommige afbeeldingen zijn door AI gegenereerd.





AI GENERATED

Wat leer je in dit hoofdstuk?

Na dit hoofdstuk kan je:

- ✓ de **drie gouden regels** van wachtwoordbeheer toepassen
- ✓ hashing, salting en rainbow tables uitleggen, en waarom **salting** beschermt
- ✓ **CRAM, SCRAM en passkeys** onderling situeren
- ✓ de vier **MFA-factoren** (weten, hebben, zijn, waar/wanneer) combineren
- ✓ **TOTP** uitleggen, en waarom passkeys veiliger zijn tegen real-time phishing

Authenticatie vs autorisatie



Authenticatie

Wie ben je?

Het bewijs van je identiteit.



Autorisatie

Wat mag je?

Het toekennen van rechten.

Dit hoofdstuk focust op **authenticatie**: hoe gaan we als beheerders veilig om met login-informatie?

DE GEHEIME SLEUTEL

Je wachtwoord is in essentie je **geheime sleutel**.

Login = gebruikersnaam + wachtwoord. Het weten van die sleutel is een eerste vorm van authenticatie. Als beheerder ga je er dus **uitermate veilig** mee om.

DEEL 1

Veilige wachtwoorden

Veel hangt af van hoe gebruikers zelf met hun wachtwoorden omgaan.

Veilige wachtwoorden: de regels

Regels die continu **niét** gehanteerd worden:



Nooit hergebruiken

Eén wachtwoord
per service.



Geen dictionary

Geen woorden uit
dictionaries.
Overweeg random
gegenereerde
wachtwoorden.



Lang genoeg

Minimum **16**
tekens.



Gemengd

Cijfers, letters (groot
én klein) en
leestekens.

Hoe stropers aan wachtwoorden komen

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



Password spraying

Veelgebruikte wachtwoorden testen op veel accounts:
“spray and pray”.



(Spear) phishing

Nep-mail of -bericht → malware of een fake login.
Spear phishing is op maat, gericht op één doelwit.



Keyloggers

Alle toetsaanslagen opnemen en later analyseren.

Spear phishing is vandaag één van **dé** social engineering aanvallen bij uitstek.

Online vs. offline aanvallen

Online

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

vs

Offline

- De aanvaller heeft (lees)toegang tot de **wachtwoord-databank**
- Via SQL injection, een insider, een datalek, ...

GA ER VAN UIT DAT HET GEBEURT

Je databank zal **vroeg of laat lekken.**

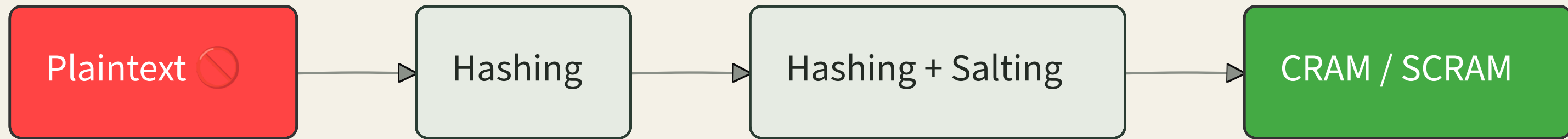
Vanaf nu sta je in de schoenen van de beheerder: hoe beperk je de schade bij zo'n lek tot een minimum?

DEEL 2

Hoe wachtwoorden opslaan

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

Van naïef naar veilig



Paswoorden als plaintext

- Databank met twee kolommen: gebruikersnaam + wachtwoord
- Het wachtwoord staat er **zoals het is**
- Databank gestolen? Dan heeft de aanvaller **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?

Mail met je **originele wachtwoord**? Dan staat het er als plaintext.

Test het met de “Ik ben m’n wachtwoord vergeten”-knop. Een service zou jouw wachtwoord **nooit** mogen kunnen zien. Uiteindelijk mag je wachtwoord zelfs nooit je computer verlaten.

FACEBOOK, 2019

Wie zou nu zo dom zijn?!

Wachtwoorden van honderden miljoenen gebruikers in **plaintext** bewaard.

200-600

miljoen gebruikers **getroffen**

~20.000

Facebook-medewerkers hadden
toegang

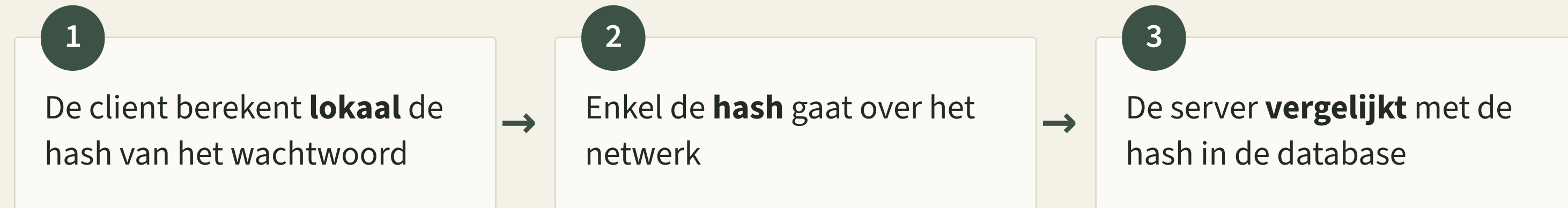
jaren

werden ze **zo bewaard**

Bron: The Verge (2019)

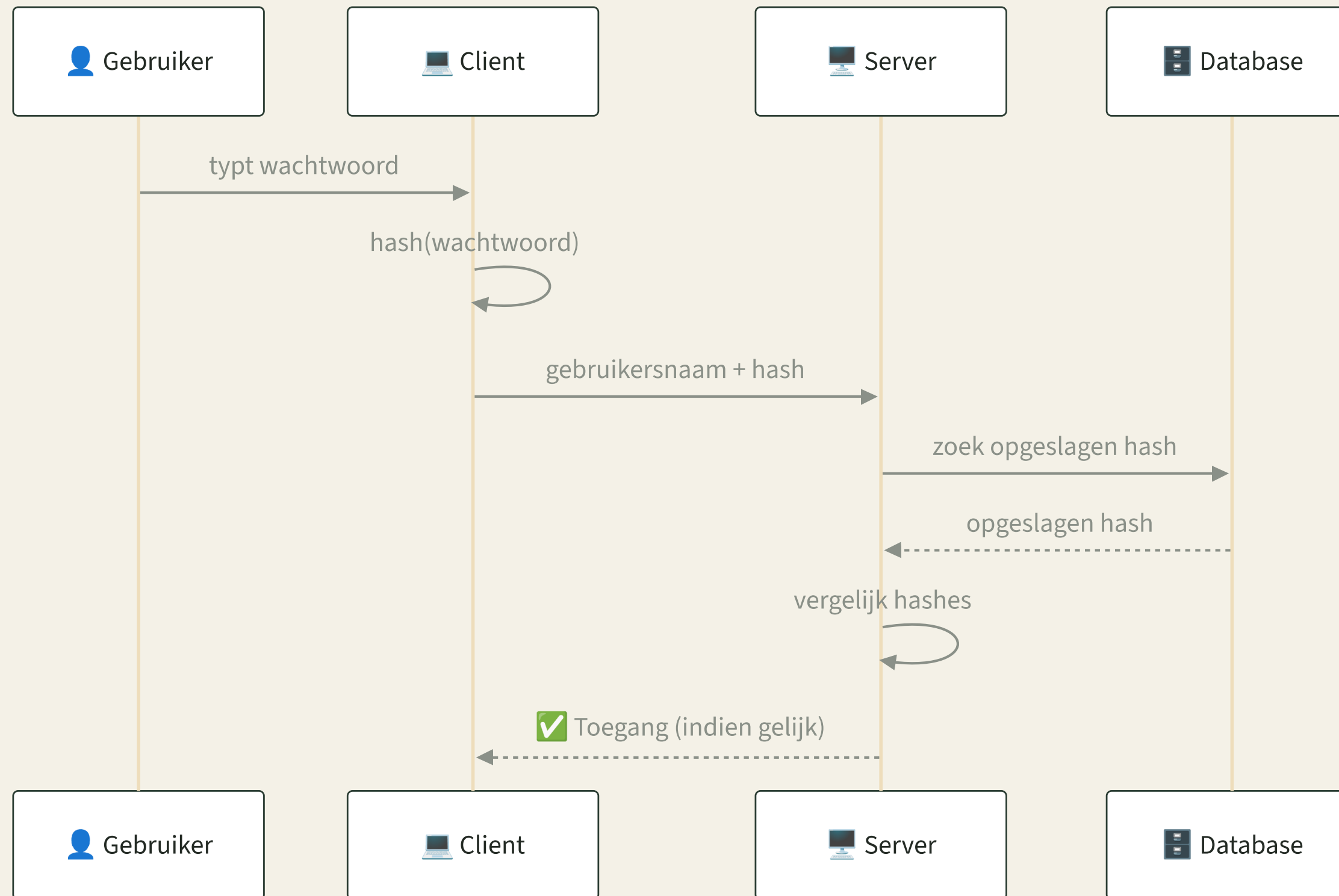
STAP 2

Paswoord hashing



Gelijk? Dan kende de gebruiker het juiste wachtwoord.

Hashing flow

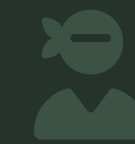
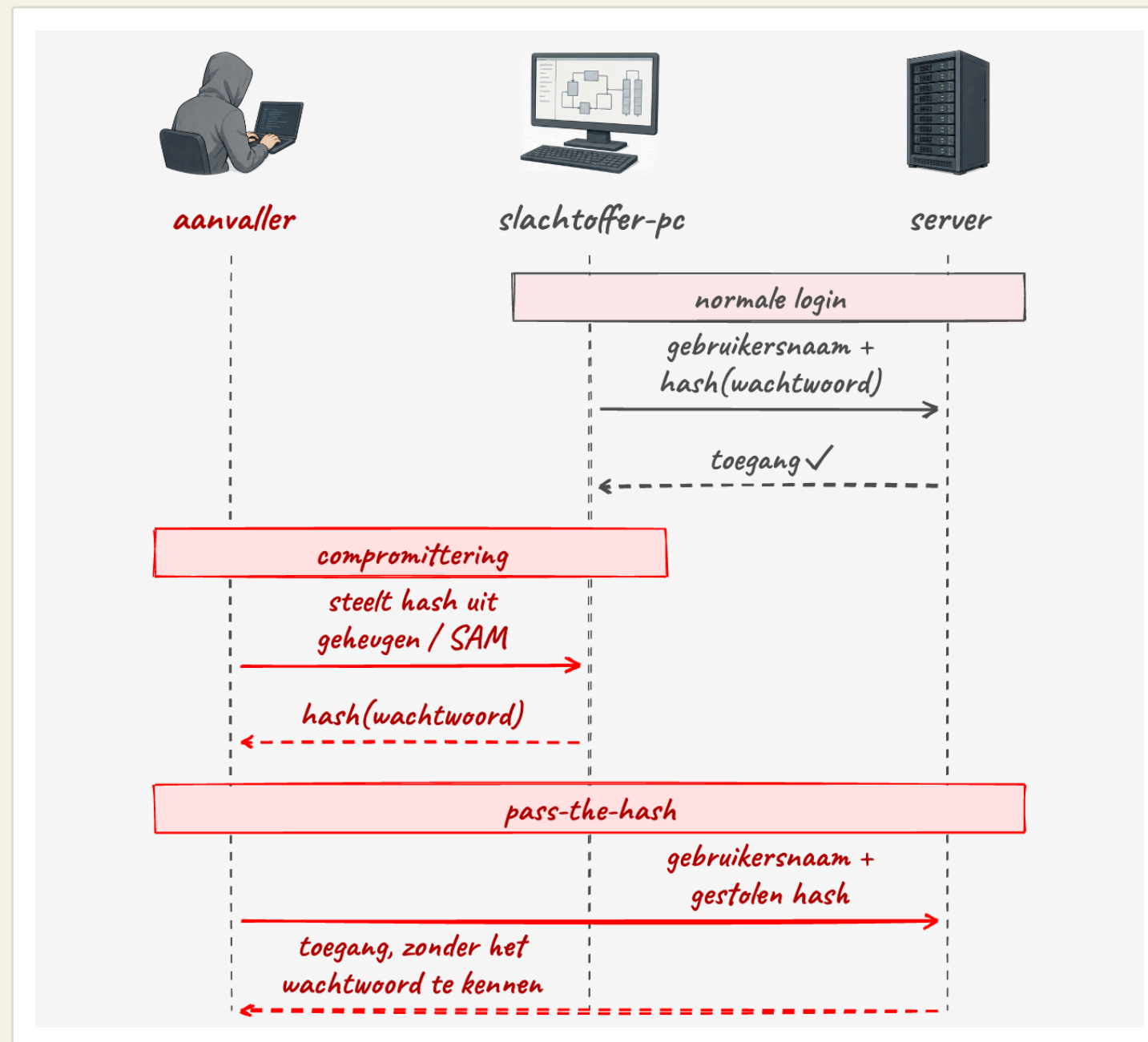


HET GEVAAR VAN HASHING

Hashing is vatbaar voor een **pass-the-hash** aanval.

Een geldige gebruikersnaam + hash capteren volstaat om in te loggen. Het originele wachtwoord kennen hoeft niet.

Pass-the-hash aanval



Geen kraakwerk nodig

De aanvaller hoeft het originele wachtwoord **nooit te kraken**: de hash zelf is genoeg om zich aan te melden.

Probleem met hashing

Een *secure hash* is **niet omkeerbaar**. Toch kan dit gebeuren:



Collisions

Twee verschillende wachtwoorden → dezelfde hash.



Hergebruik

Twee gebruikers met hetzelfde wachtwoord → dezelfde hash.

Collisions zijn bij moderne hashes zeldzaam. Voor hergebruik is de oplossing **salting** (zie verder).

Veel websites hashen wel **aan serverzijde**, maar dan over een beveiligde **TLS-tunnel**.

Rainbow table attack: het dilemma

Databank gestolen? Dan moet de aanvaller de hashes nog kraken.

Ter plekke hashen

- *bcrypt*, *scrypt* en *pbkdf2* zijn **by design traag**
- → **timebottleneck**

vs

Vooraf precomputen

- Alle mogelijke hashes op voorhand berekenen
- Die lijst wordt gigantisch → **memorybottleneck**

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

8 KLEINE LETTERS

Even concreet

$2 \cdot 10^{11}$

mogelijke wachtwoorden (26^8)

2,4 dagen

bruteforce aan **1 miljoen** hashes per seconde

1,46 TB

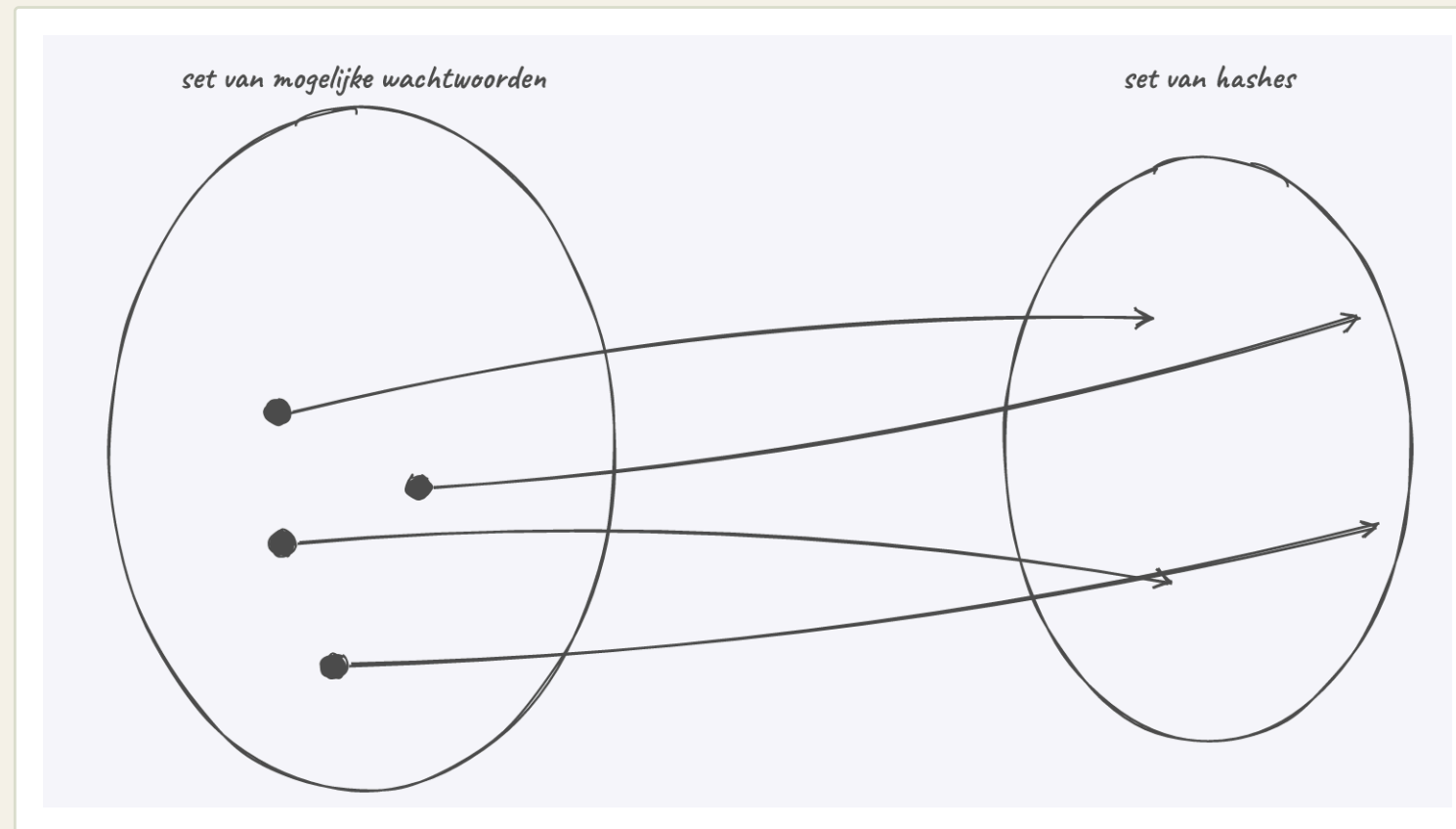
om **alle hashes** vooraf op te slaan

~1 minuut

met een GPU aan **2,8 miljard** per seconde (2011)

Puur bruteforcen *of* puur precomputen loont nauwelijks.

Rainbow table: het concept



Zonder rainbow table: ieder wachtwoord mapt naar exact één hash (omslachtig).

Een rainbow table is een **gecomprimeerde** tabel van precomputed hashes: niet alles bewaren, en toch een grote set bestrijken.

Vergelijk: de oneven getallen van 1 tot 101 bewaren als

`start=1, eind=101, stap=2`

Rainbow table: opbouwen

1

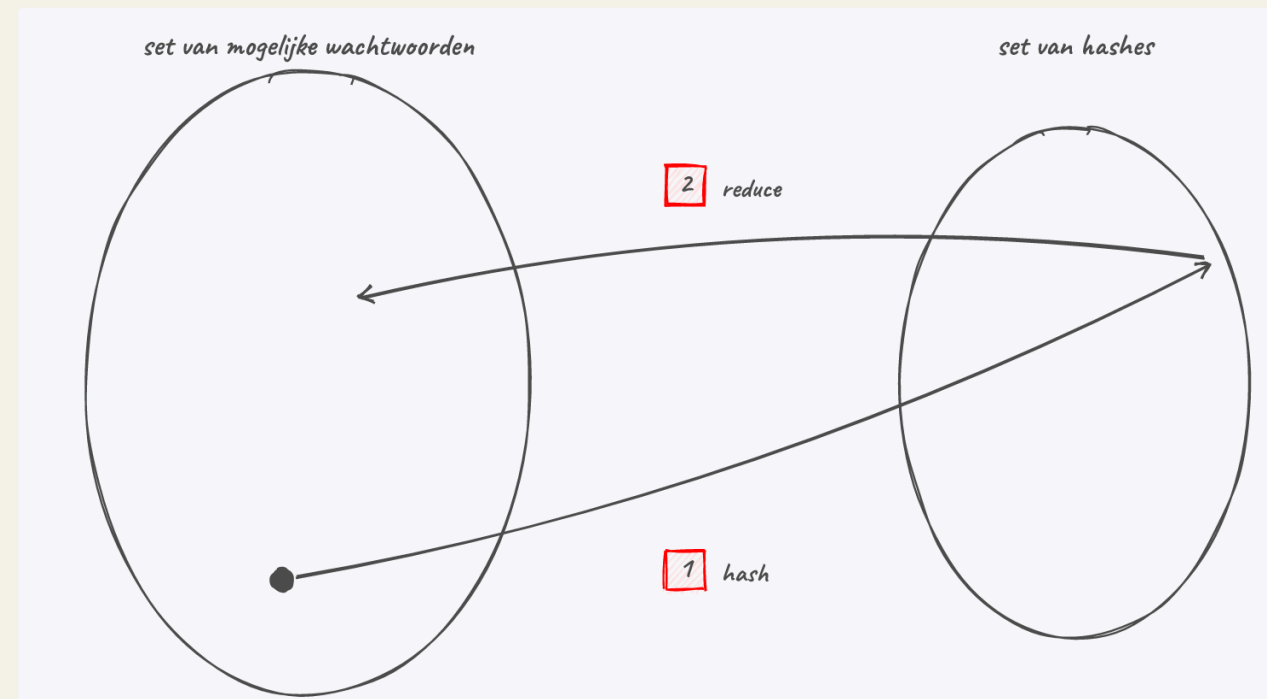
Kies een **startpunt**: een wachtwoord uit de set

2

Genereer de **hash**

3

Pas een **reductiefunctie** toe
→ terug naar een wachtwoord

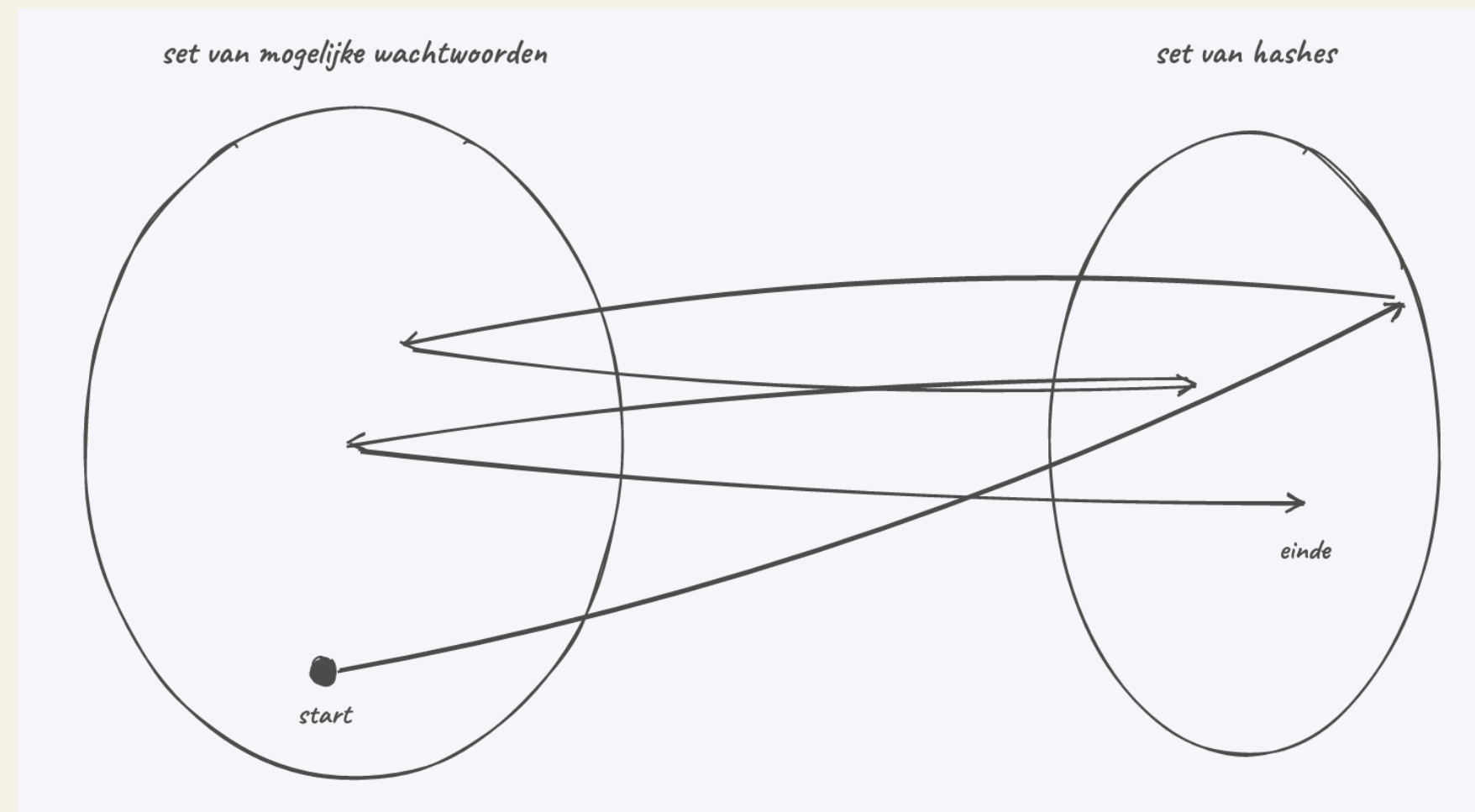


Van de hash via de reductiefunctie terug naar een ander wachtwoord.

Reductiefunctie: bv. som van de ASCII-waarden → index in de set.

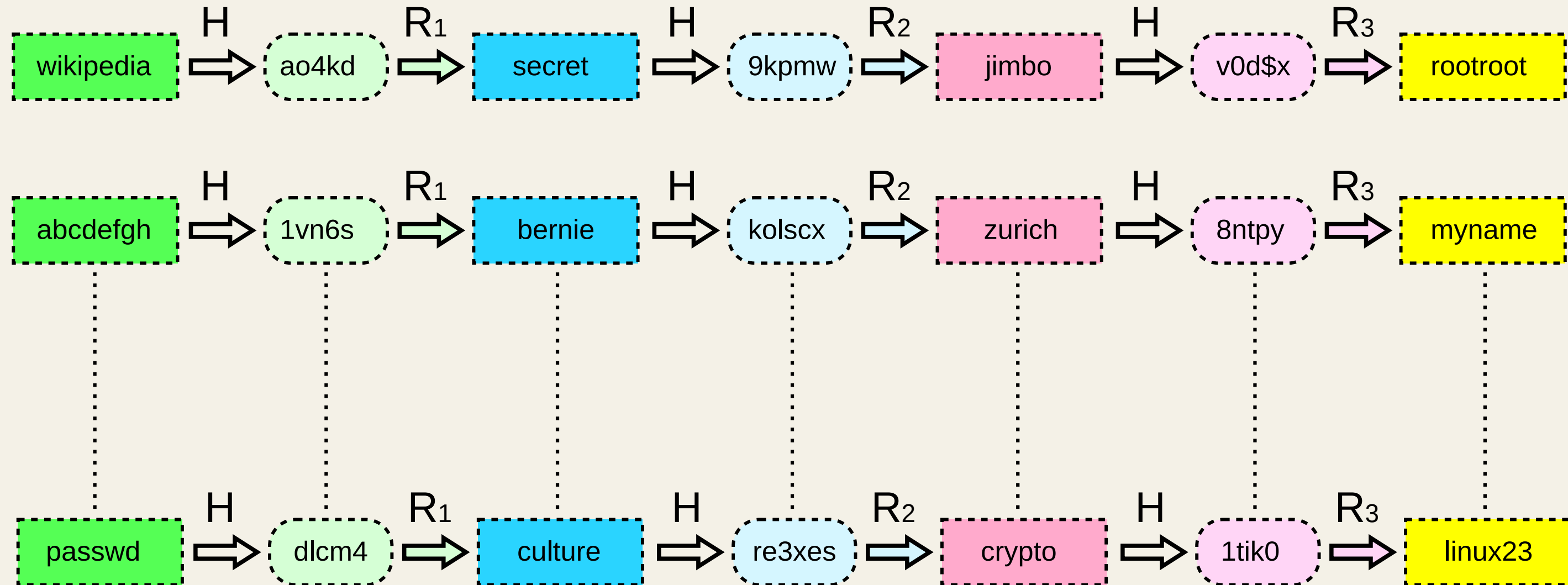
Rainbow table: de chain

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



Voorbeeld van een chain.

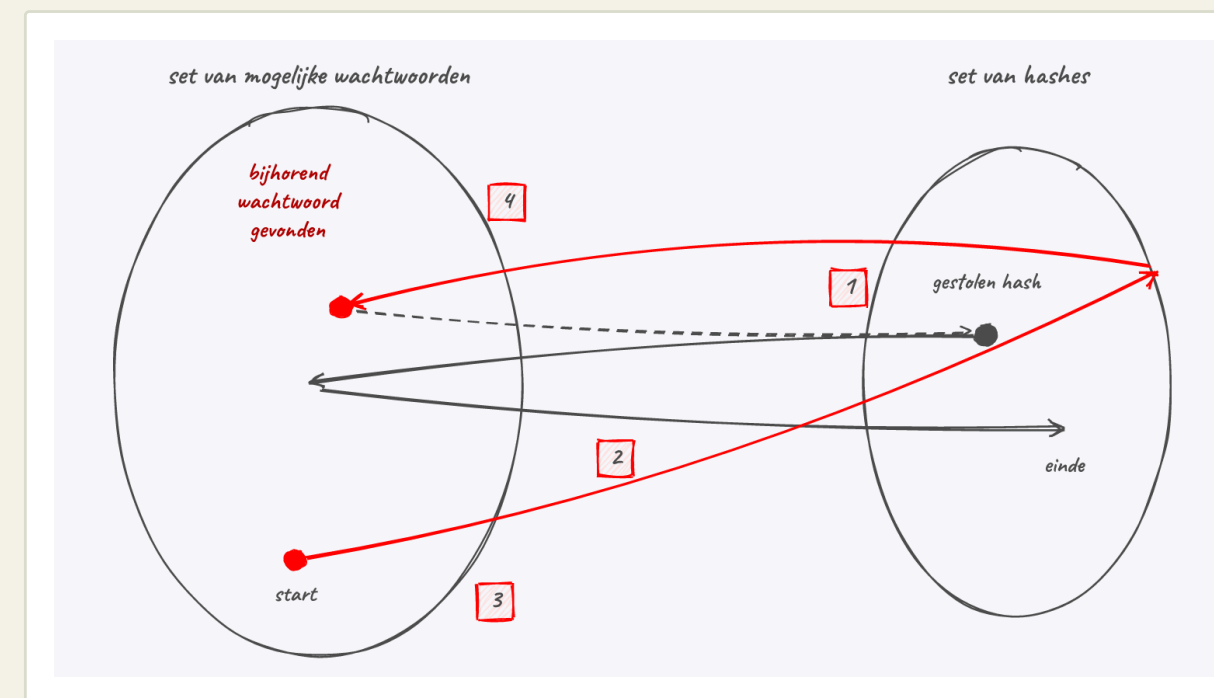
Meerdere chains met verschillende reducties



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

Rainbow table: kraken

0. Start met de **gestolen hash** (waarvan we het wachtwoord willen weten)
1. Pas reductie + hash toe tot je een **bekend eindpunt** vindt (zo weet je in welke chain je zit)
2. Neem het **startpunt** van die chain
3. Herhaal hash + reductie tot je de gestolen hash bereikt
4. Eén stap terug = het **wachtwoord**!



Het kraken van een hash met een rainbow table.

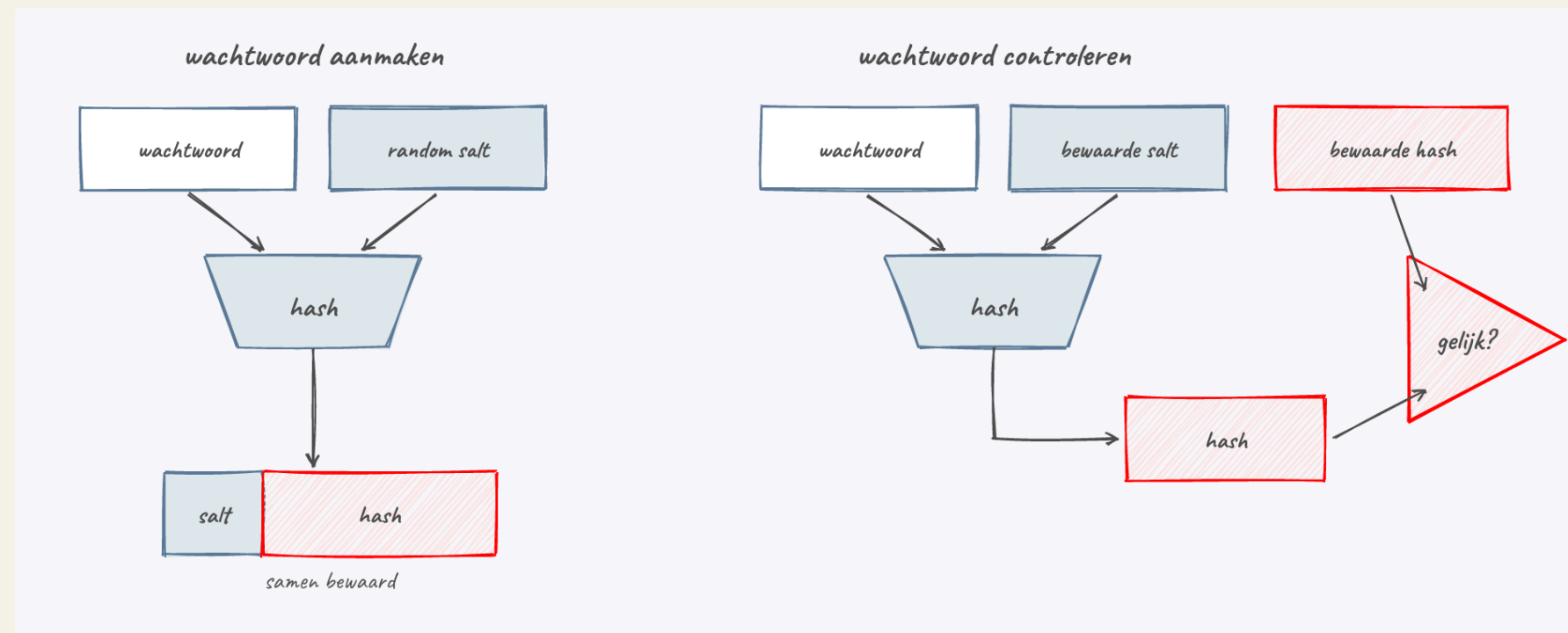
Rainbow tables: conclusie

- ✓ Een **efficiënte** manier om grote sets wachtwoorden te kraken
- ✓ Een compromis tussen tijd en opslag
- ✗ Werkt niet meer zodra er een **salt** gebruikt wordt (zie verder)

Wat als identieke wachtwoorden **verschillende hashes** kregen? → **Salting!**

Salting

Een **salt** is een random extra stuk dat je **vóór** het hashen aan het wachtwoord toevoegt.



Het salting proces.

Unieke salt per gebruiker: identieke wachtwoorden geven **andere hashes**. De set wordt te groot voor rainbow tables.

Salting: de databank

Gebruikersnaam	identificatie
Salt	random waarde (minimum 32 bits)
Hash	hash van wachtwoord + salt



Zoals een IV

Net als de initialisatievector (IV) bij WEP: zelfde sleutel, en toch telkens een andere seed. Vooraf berekende rainbow tables worden zo **waardeloos**. Een salt is wel **geen geheim**: gebruik dus ook een **trage** hash (Argon2, bcrypt, ...).



Pass-the-hash?

Ook nu ben je nog steeds **niet beschermd** tegen pass-the-hash aanvallen.

Mimikatz

Origineel een demo die aantoonde dat Microsoft-authenticatie **onveilig** was. Snel overgenomen door digitale stropers: vindt en hergebruikt *authentication tickets* op Windows.



Pass-the-hash

Windows NTLM
hashes



Pass-the-ticket

Kerberos tickets



Golden Ticket

Hash van het
KRBtgt-account
gestolen? Dan
vervals je zelf
tickets met domain
admin rechten.



Pass-the-cache

Tickets op Mac,
Unix en Linux

2017

Mimikatz en NotPetya

Mimikatz is **automatiseerbaar**, en dus gevaarlijk op schaal.



Getraceerd naar `Sandworm`, een Russische hackinggroep onder de `GRU` (militaire inlichtingendienst).

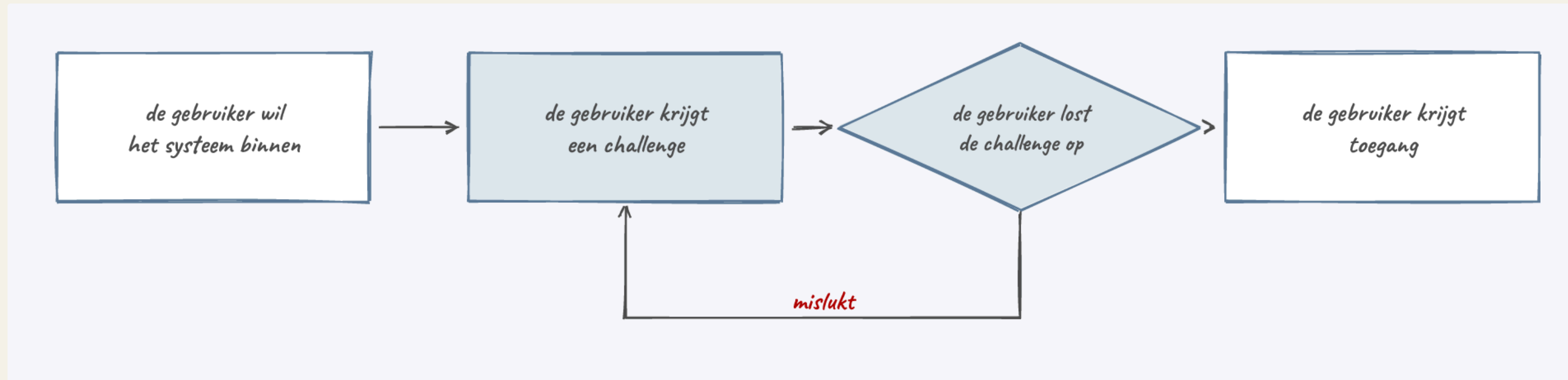
DEEL 3

CRAM en SCRAM

Challenge-Response Authentication Mechanism

CRAM

Je krijgt een **challenge** (vraag) en moet een geldig **response** (antwoord) geven.

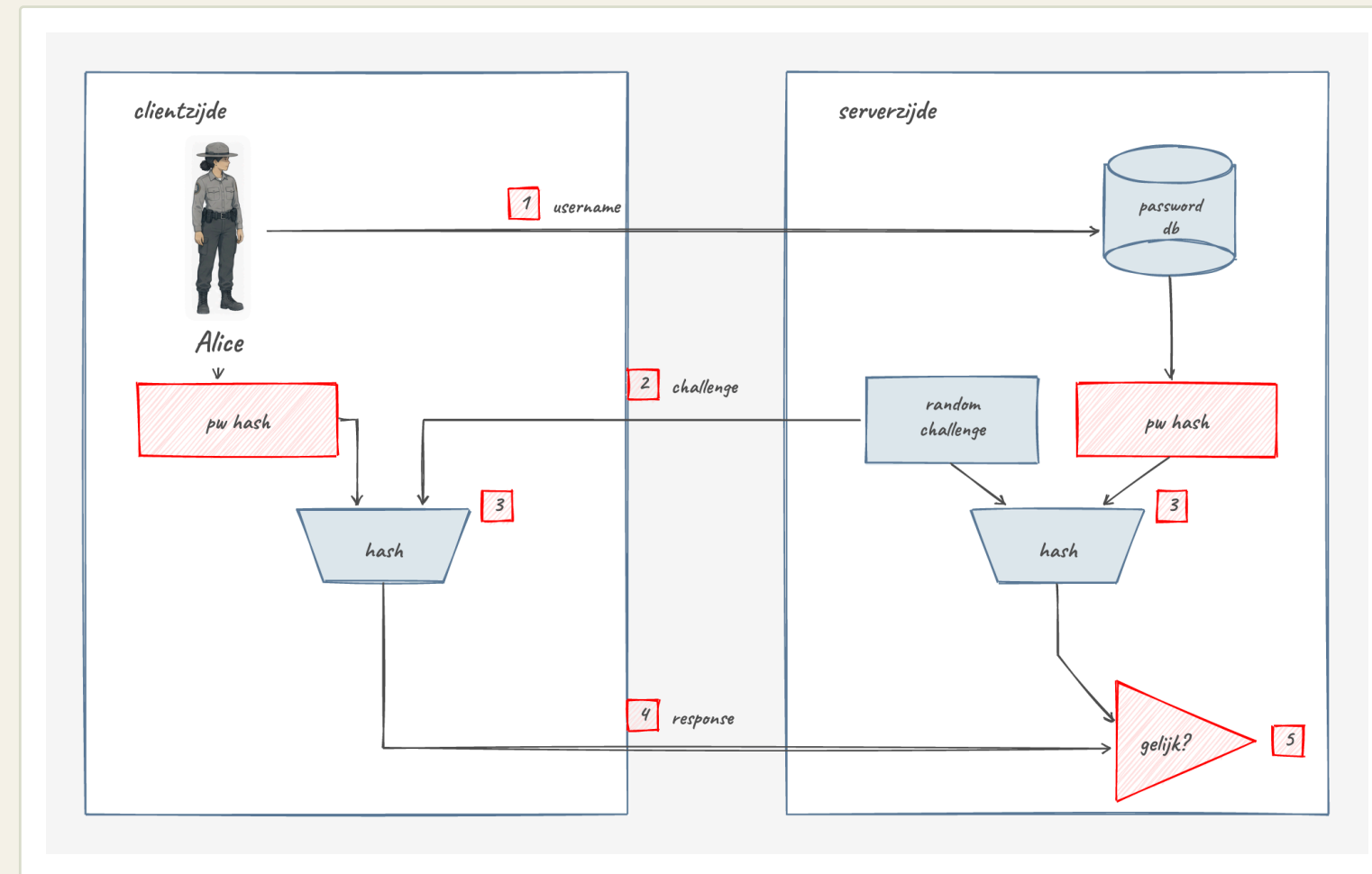


CRAM.

Voorbeelden: wachtwoord-login CAPTCHA irisscan

CRAM bij een password login

1. Client → **username**
2. Server → random **challenge**
3. Beiden berekenen
hash(challenge + password
hash)
4. Client → **response**
5. Server vergelijkt: match =
toegang



CRAM flow.

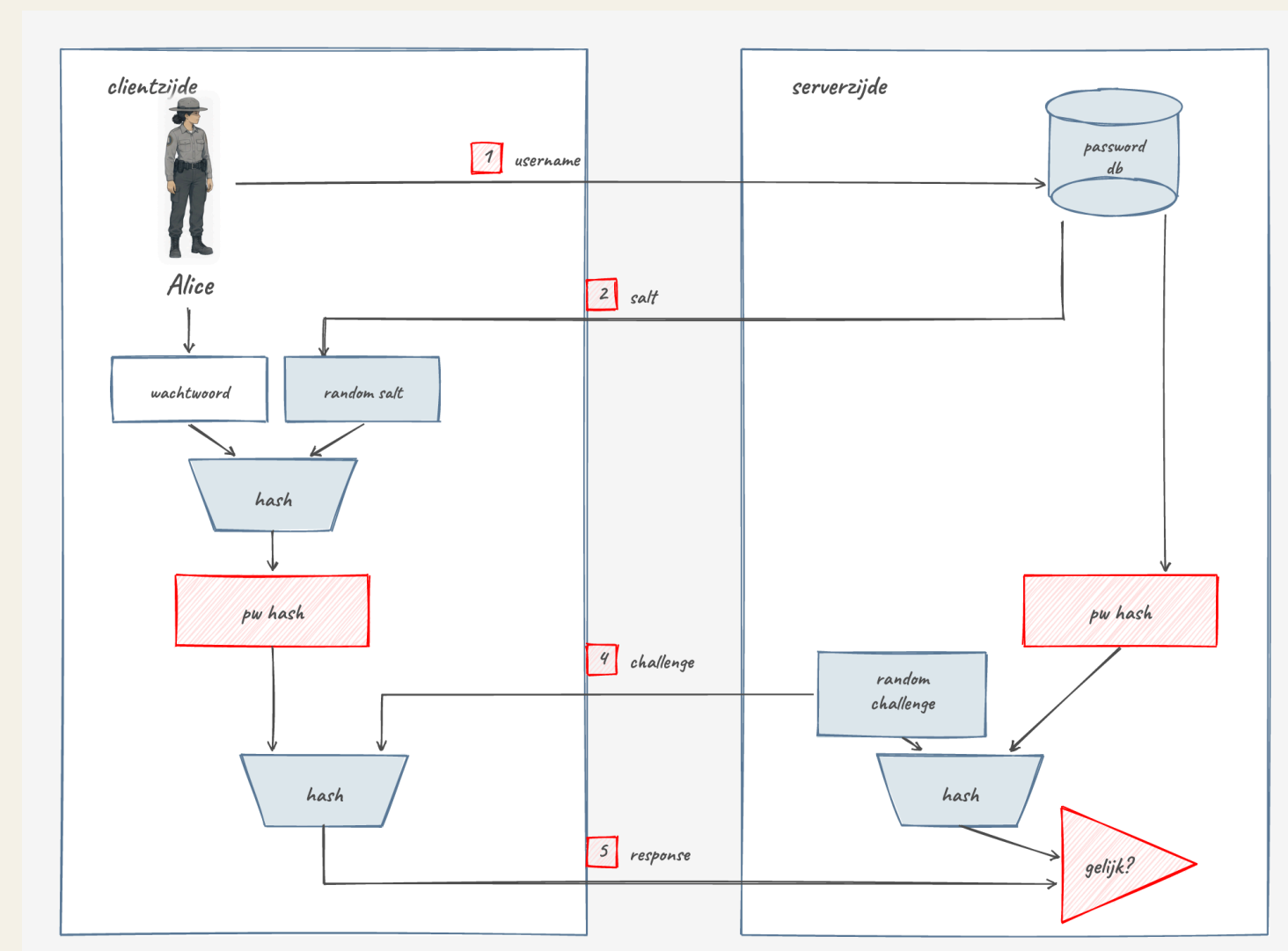
SALTED CHALLENGE-RESPONSE AUTHENTICATION MECHANISM

SCRAM

Voegt **salting** toe aan CRAM:

- ✓ De gesalte hash wordt **nooit** verzonden
- ✓ Salt bemoeilijkt offline brute-force en rainbow tables

De server stuurt de **bewaarde salt** mee. De client leidt telkens zelf de hash af uit wachtwoord + salt.



SCRAM en pass-the-hash

⚠️ Onze vereenvoudigde SCRAM

- De server bewaart de gesalte hash
- Gelekt? Dan berekent de aanvaller:
`H(gestolen_hash + challenge)`
- **Geen** bescherming tegen pass-the-hash

vs

🛡️ Echte SCRAM (RFC 5802)

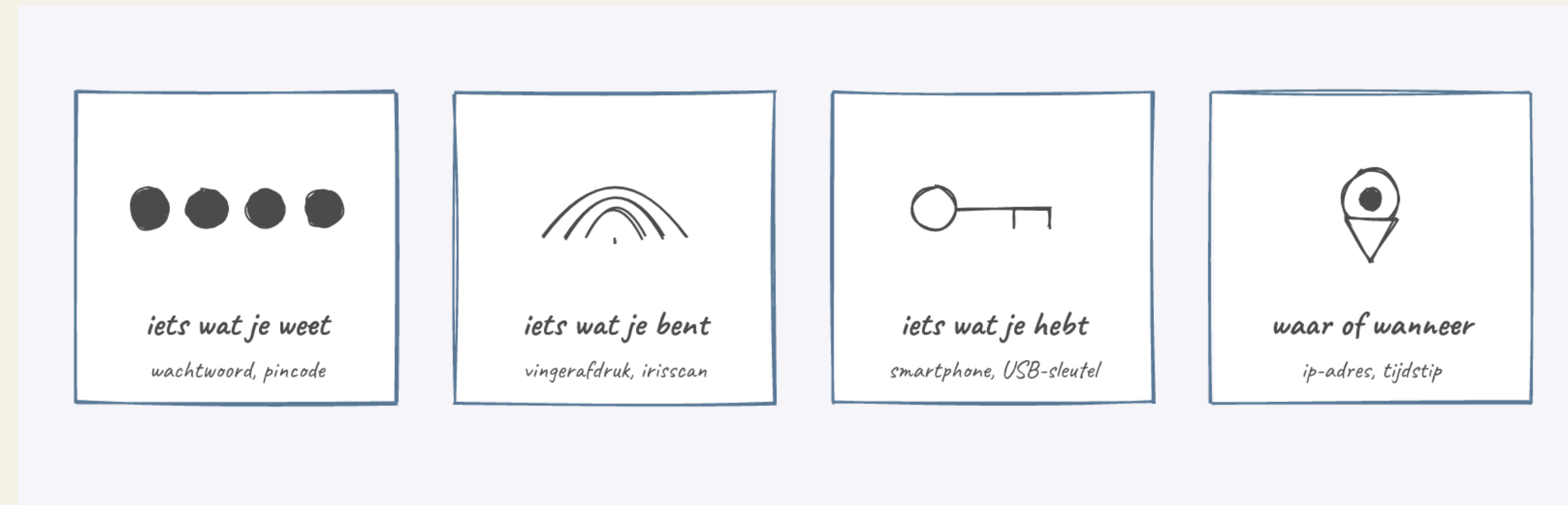
- De server bewaart enkel
`StoredKey = H(ClientKey)`
- Uit `StoredKey` kan je `ClientKey` niet afleiden
- Pass-the-hash wordt **tegengehouden** (tenzij de aanvaller ook een login afluistert)

DEEL 4

Multifactor authentication

Er is meer dan enkel wachtwoorden.

Multifactor authentication (MFA)



Iets wat je weet

Wachtwoord,
pincode,
rijksregisternummer

Iets wat je bent

Vingerafdruk,
irisscan (*biometrics*)

Iets wat je hebt

Smartphone, USB-
key

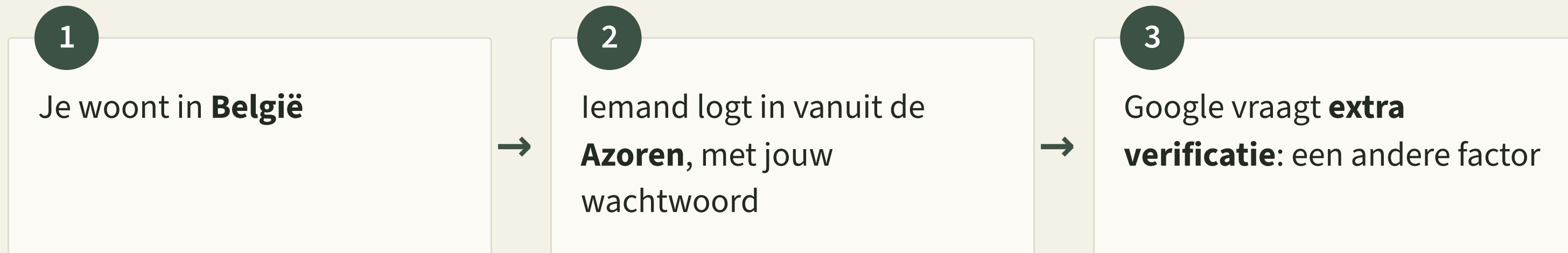
Waar/wanneer

IP-adres, tijdstip
van login

MFA in de praktijk

MFA = minstens twee factoren, **2FA** = precies twee. Meer factoren is veiliger, maar ook minder gebruiksvriendelijk.

Systemen passen MFA **dynamisch** aan:



Iets wat je weet

Het grote probleem met *dingen weten*:



Vergeten

Je kan ze vergeten, en dan kan je niet meer inloggen.



Uitlekken

Anderen kunnen ze te weten komen en zich als jou voordoen.

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

FACTOR 2: IETS WAT JE BENT

We zijn **allemaal uniek**.

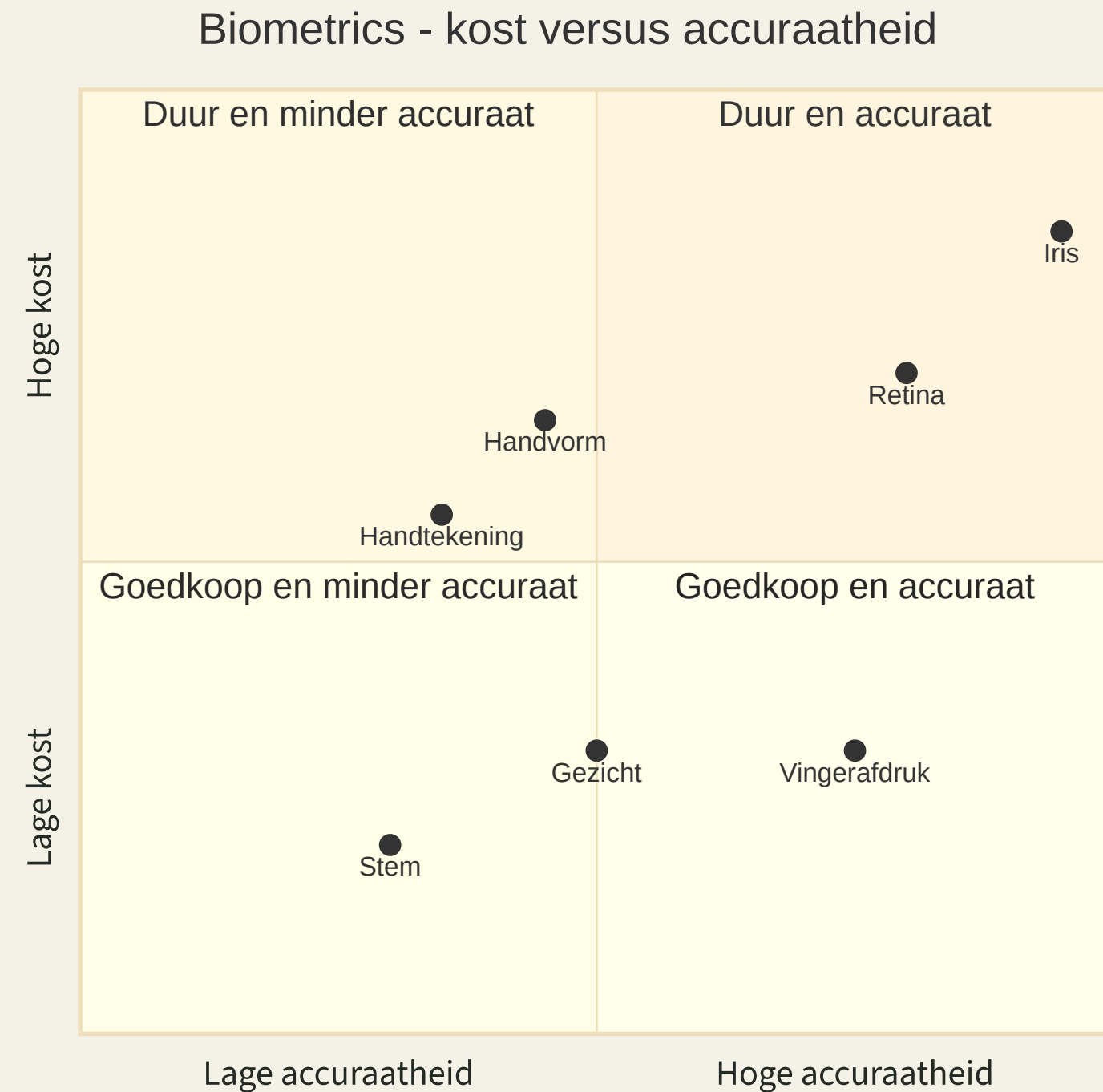
Veelgebruikte biometrieën: `vingerafdruk` `iris` `stem`

`gezicht (3D stereo camera)`

Maar ook: `manier van typen` `manier van wandelen (gait)`

Biometrics: kost vs. accuraatheid

Keuze hangt af van **use case** en **budget**: irisscan aan de grens $\neq$ gezichtsherkenning via webcam.



INDIA

Biometrics en privacy

Biometrische gegevens opslaan maakt **privacy** een erg heikel punt.

1,1 miljard

burgers in de **Aadhaar**-
databank

10

vingerafdrukken per burger,
plus de **irisscan**

2018

het jaar waarin de databank
gehackt werd

Biometrics: opslag

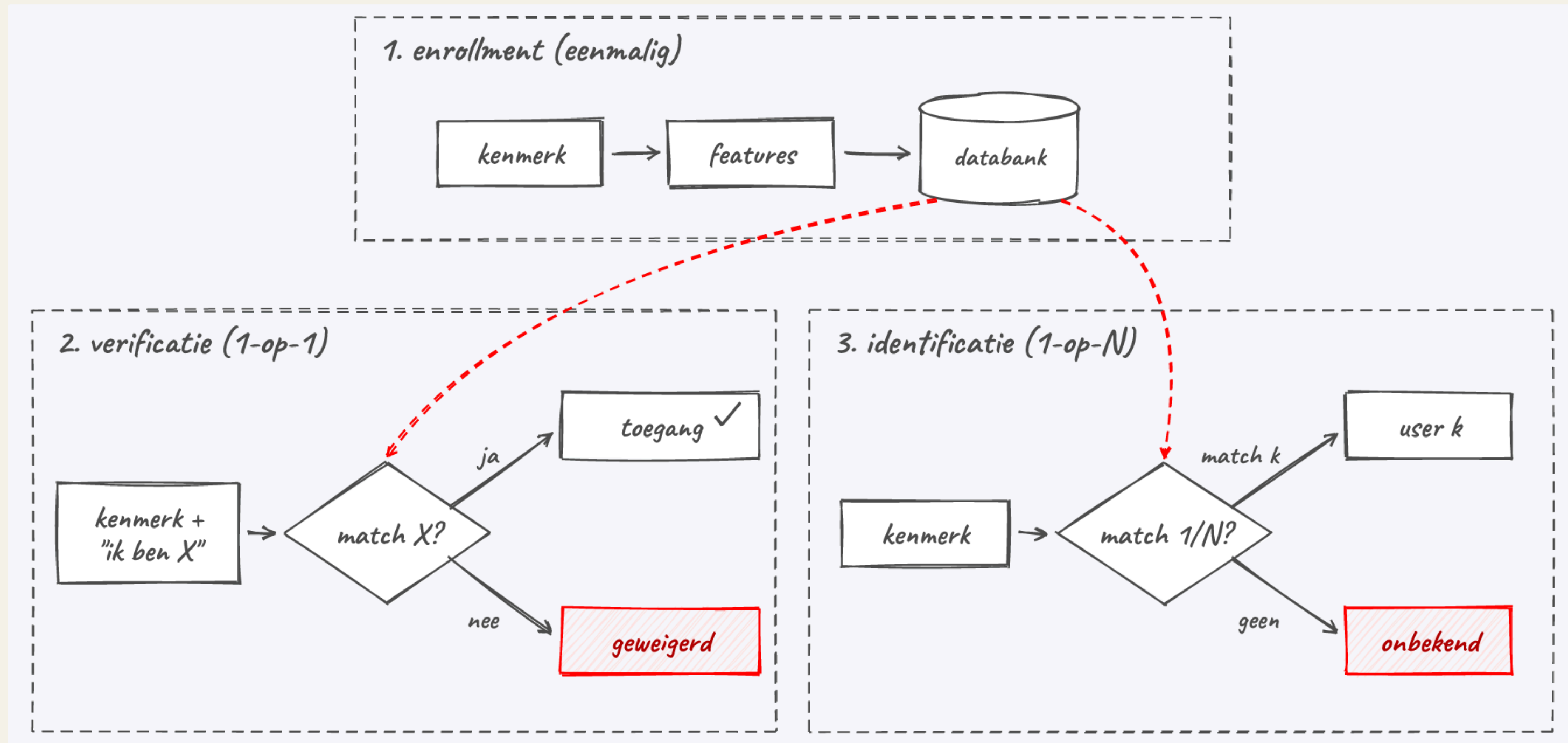
Feature points	unieke waarden per persoon
-----------------------	----------------------------

In de databank	een korte reeks getallen
-----------------------	--------------------------

Bij login	(quasi) dezelfde feature points → match
------------------	--

“Quasi”, want registreren is nooit 100% accuraat: denk aan een krasje op je vinger of een andere baard.

Biometrics: drie fasen



Verificatie of identificatie?

Verificatie (1-op-1)

- “Ben jij wel degelijk user X?”
- Vergelijking met **één** template
- Het klassieke inlogscenario

vs

Identificatie (1-op-N)

- “Wie ben jij?”
- Vergelijking met **elke** template in de databank
- Vél zwaarder: grenscontrole, forensisch onderzoek

Biometrics als identificatie: een aanvaller kan niet meer zeggen “*ik ben persoon X*” terwijl de scanner de vingerafdruk van persoon Y registreert.

Iets wat je hebt

Een fysiek object is **moeilijk na te maken**. In essentie bevat het een veel langer wachtwoord dan een mens kan onthouden.



Smartphone

Met een authenticator app
(Google Authenticator, ...)



USB-sleutel

YubiKey, Titan Key, ...



Nadeel

Een fysiek object kan je
verliezen, of het gaat **stuk**.

DEEL 5

Federation en SSO

Zelf logindata beheren? Grote **kans op fouten**.

Single Sign-On (SSO)

Eénmaal inloggen → toegang tot **meerdere** applicaties.

Inloggen bij Google = meteen Gmail YouTube Drive ...



Identity provider (IdP)

De centrale partij waaraan de authenticatie wordt uitbesteed. Bevestigt wie je bent.

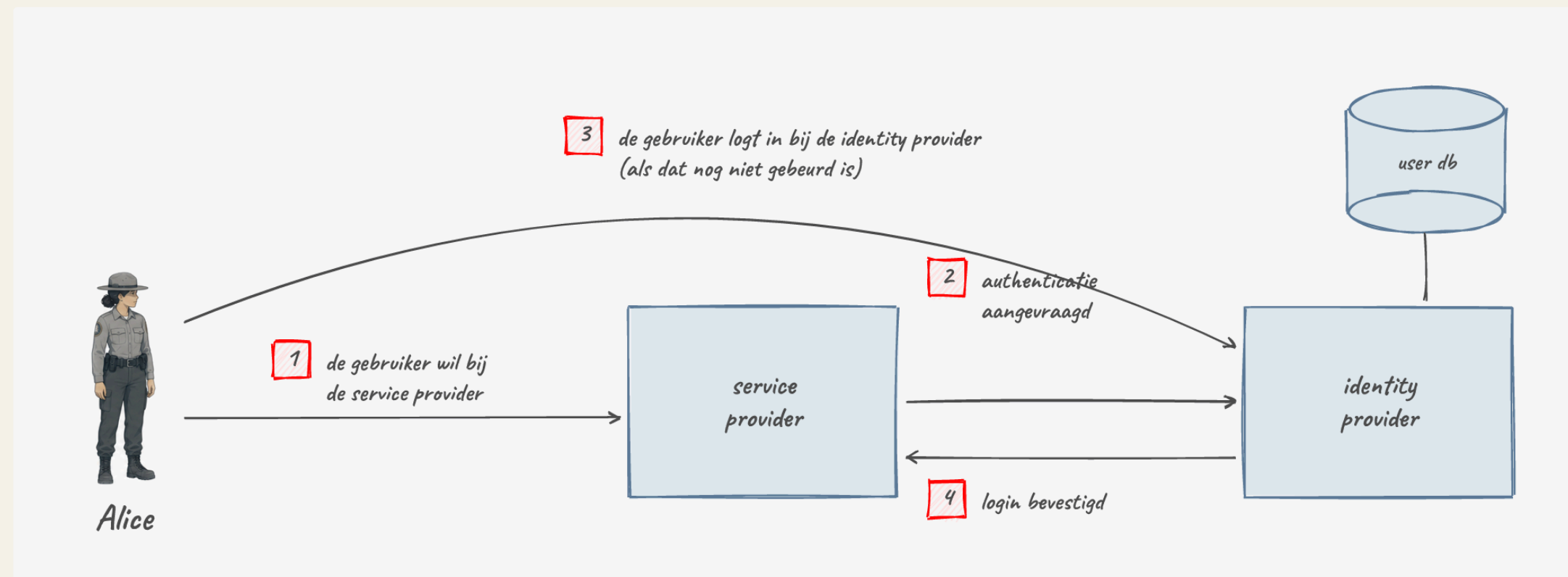


Service provider

De applicatie. Vertrouwt op de bevestiging van de IdP.

Federation

SSO over de grenzen van organisaties heen: inloggen met je bestaande Google-, Facebook-, ... account.



Een vereenvoudigd SSO proces.

Jouw site = **service provider**, Google/Facebook = **identity provider**. Hoe je gebruikerswachtwoorden opslaat, is dan niet langer jouw zorg.

Delegation vs federation

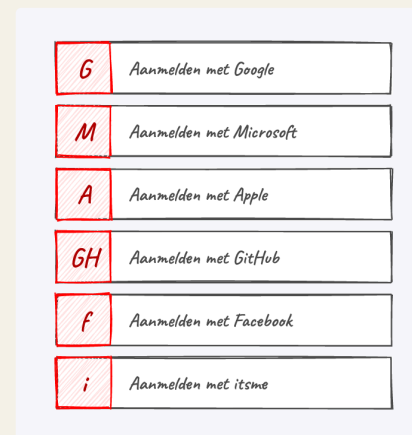
Delegation

De website laat inloggen over aan **één specifieke** third-party (bv. enkel Facebook).

vs

Federation

Éénder welke identity provider die dezelfde standaard spreekt (bv. **OpenID Connect**).



Typische SSO knoppen.

OAuth: een app toegang geven tot je gegevens bij een andere dienst (**autorisatie**).

OpenID Connect bouwt erop verder voor het inloggen zelf (**authenticatie**). In bedrijven ook: **SAML**.

PRIVACY

In hoeverre **vertrouw** je Google of Meta met
jouw (login)data?

Bij federatie doet een third-party de eigenlijke authenticatie.

DEEL 6

WebAuthn en passkeys

Inloggen **zonder wachtwoord**.

PASSKEYS

Een passkey is een **sleutelpaar** dat je toestel per website aanmaakt. Het combineert **asymmetrische crypto** met **hardware authenticatie**.

Met **WebAuthn** log je in zonder wachtwoord, via een **authenticator** die je bij je hebt: een USB-sleutel of je smartphone. Geen complexe wachtwoorden meer onthouden.

Termen die je zal tegenkomen

FIDO Alliance	organisatie achter de standaarden (<i>Fast IDentity Online</i>)
----------------------	---

WebAuthn	Web Authentication API, laat websites met authenticators communiceren
-----------------	---

FIDO2	overkoepelende standaard: WebAuthn (website ↔ browser) + CTAP (browser ↔ authenticator)
--------------	---

U2F	Universal 2nd Factor, een oudere standaard
------------	--

▶ Duidelijke uitleg over passkeys: youtu.be/cEhc6vMFTh4

Passkey registratie

1

Je kiest een **passkey** in plaats van een wachtwoord



2

Je identificeert je **lokaal**:
vingerafdruk, PIN, YubiKey, ...



3

Je toestel maakt een **sleutelpaar**: de private sleutel blijft, de publieke gaat naar de website

De website heeft **nooit** toegang tot de private sleutel. “*Het wachtwoord verlaat nooit het apparaat.*”

Het attestation object

De publieke sleutel gaat niet *zomaar* over de draad, maar zit verpakt in een **attestation object**:



Publieke sleutel

Die de website bewaart.



Signed challenge

De challenge van de server, als bewijs van authenticiteit.



Credential ID



Certificaat

Bewijst welk **type authenticator** de sleutel maakte (bv. een echte YubiKey), niet wie jij bent.

SYNCED PASSKEYS

Verlies je je authenticator, dan ben je **al je accounts kwijt.**

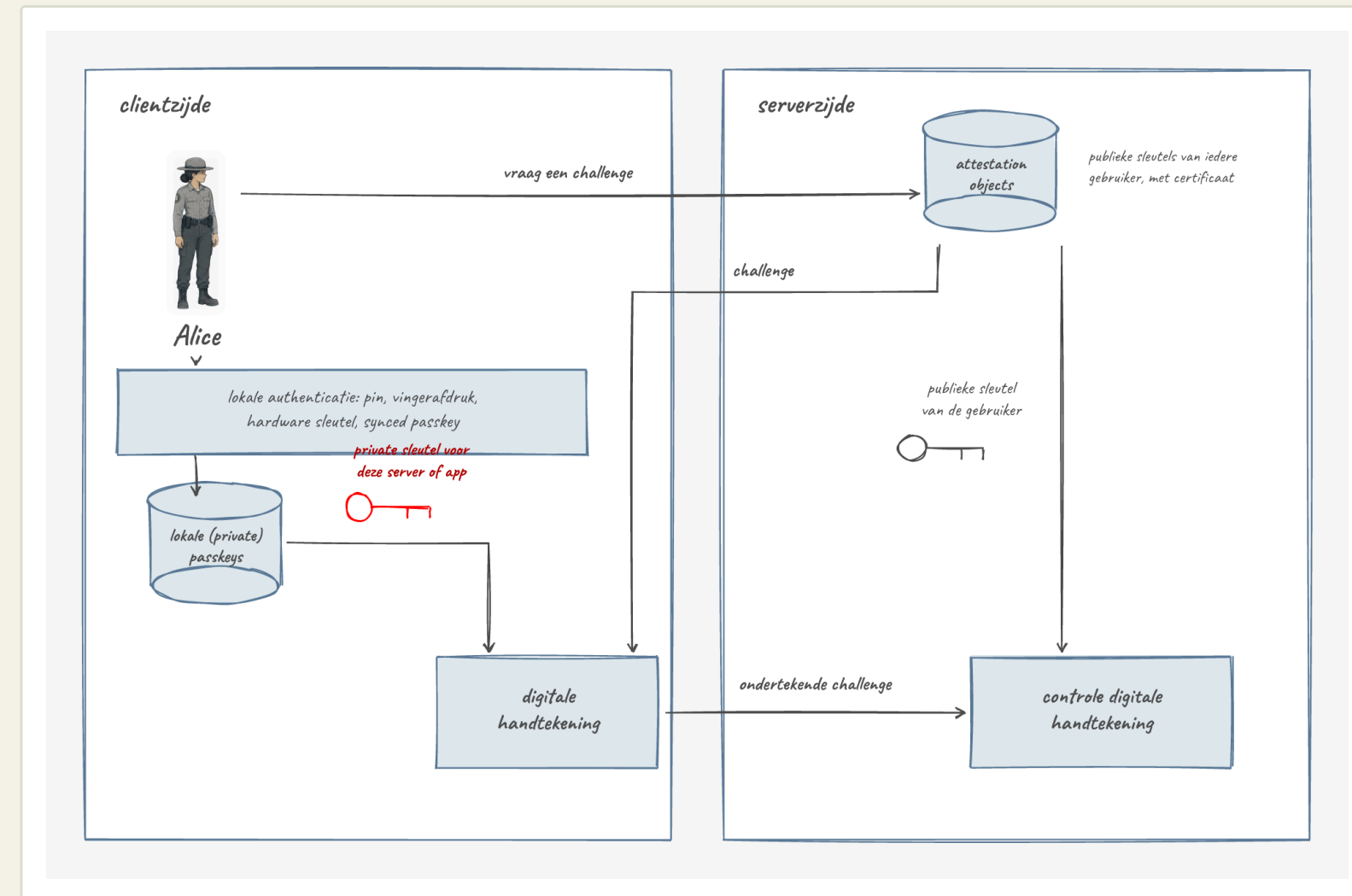
Je private sleutels staan op de authenticator. Een **password manager** met **synced passkeys** bewaart ze veilig én synchroniseert ze naar je andere apparaten.

Inloggen met een passkey

Op basis van **public key crypto**:

1. Server stuurt een **challenge**
2. Jouw toestel **ondertekent** die met je **private sleutel**
3. Server verifieert de handtekening met je publieke sleutel
4. Klopt het? **Geauthenticeerd**

De passkey hoort bij **één domein**:
een phishingsite krijgt geen handtekening.



Het login proces met een passkey.

DEEL 7

Authenticator apps en TOTP

Time-based One-Time Password

Authenticator apps

Google Authenticator

Microsoft Authenticator

Authy

30 s

en je krijgt een **nieuwe code**

6

cijfers per code, in te voeren naast je wachtwoord

Hoe maakt een app op jouw telefoon dezelfde code als de server verwacht? Met **TOTP**: *Time-based One-Time Password*.

TOTP: de gedeelde geheime sleutel

Bij het koppelen van je authenticator app:



Dit is het **enige moment** waarop de sleutel wordt uitgewisseld. Daarna communiceren app en server nooit meer rechtstreeks.

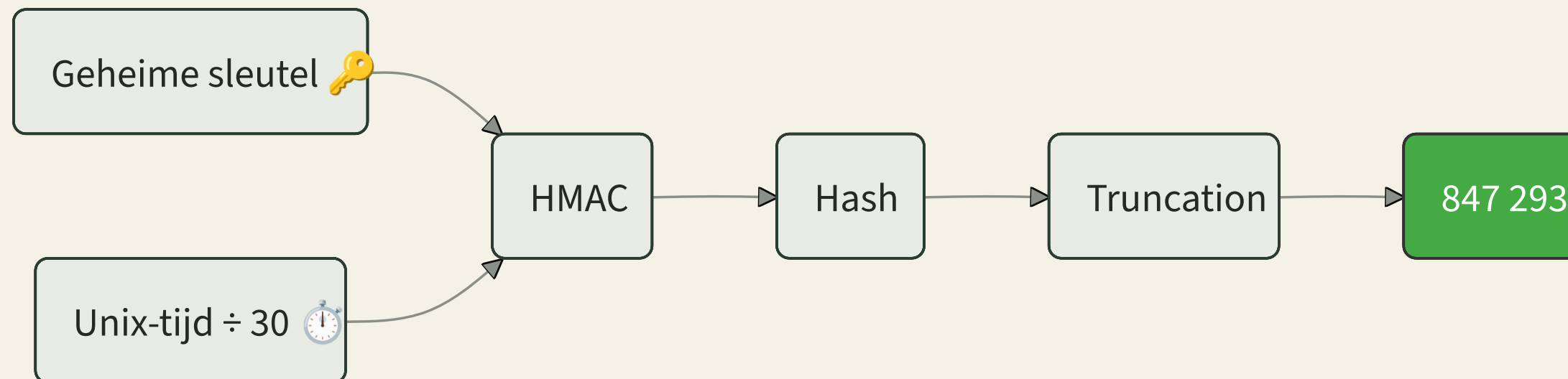
Waarschuwing

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

TOTP: het algoritme

Twee ingrediënten: de **gedeelde sleutel** en de **huidige tijd**.

Tijdstap	Unix-tijd (seconden sinds 1 jan 1970) ÷ 30, naar beneden afgerond
HMAC	geheime sleutel + tijdstap → een lange hash
Dynamic truncation	hash → een zescijferig getal : de code op je scherm



TOTP: waarom is het veilig?



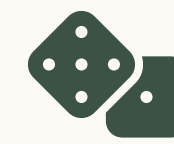
Eenmalig

Een code is slechts **30 seconden** geldig. Een onderschepte code is snel vervallen.



Offline

Na de registratie is **geen internet** meer nodig. Enkel de tijd synchroniseert.



Onvoorspelbaar

Zonder de geheime sleutel zijn toekomstige codes **onmogelijk** te berekenen.

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

TOTP is niet onfeilbaar

Real-time phishing: een **proxy** tussen jou en de echte website.

TOTP

- De proxy onderschept wachtwoord **én** TOTP-code tegelijk
- De aanvaller kan nog steeds inloggen

vs

Passkeys

- Gebonden aan het **domein** van de website
- Hiertegen beter beschermd

DEEL 8

Zero trust

Never trust, always verify.

Van burcht naar zero trust

De burcht

- Eén dikke muur: de **firewall**
- Wie binnen is (of via VPN binnenkomt), wordt **vertrouwd**
- “*No More Chewy Centers*”: harde buitenkant, zachte kern (Kindervag, 2010)

vs

Zero trust

- Er is geen “binnen” meer: cloud, thuiswerk, smartphones
- Stropers zitten vaak al **maanden** binnen (dwell time)
- Geen vertrouwen op basis van het netwerk waar je zit

Ieder verzoek, telkens opnieuw



Wie ben je?

Sterke authenticatie: **MFA** of een passkey.



Welk toestel?

Van het bedrijf? Up-to-date?



Welke context?

Vanwaar, op welk uur? (cfr. de login vanop de Azoren)



Wat mag je?

Enkel die ene applicatie: *least privilege*.

Laten alle applicaties hun login via één **identity provider** (SSO) lopen, dan is dat meteen de plek om die controles af te dwingen.

Zero trust in de praktijk

2009: Operation Aurora



Google BeyondCorp: geen VPN meer



2020: NIST SP 800-207

Zero trust is **geen product** dat je koopt, maar een manier van denken die je stap voor stap invoert. Hoe je het in een netwerk bouwt, zie je in **Network Security**.

Drie gouden regels

Alles in dit hoofdstuk valt terug te brengen tot:



Client → server

Het wachtwoord reist **nooit leesbaar** van de client naar de server.

TLS

CRAM

SCRAM



Server → client

Het wachtwoord gaat **nooit** van de server naar de client.

wachtwoord vergeten-test



Op de server

Het wachtwoord staat er **nooit** in leesbare vorm.

trage hash + salting

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

Conclusie

Authenticatie

Bewijzen wie je bent.

Autorisatie = wat je mag.

Wachtwoorden

Nooit plaintext. Een **trage wachtwoord-hash** (Argon2, bcrypt) + **salting** als minimum.

CRAM / SCRAM

Het wachtwoord gaat nooit over de draad. De échte SCRAM stopt ook pass-the-hash.

MFA

Weten, zijn, hebben, waar/wanneer. Meer factoren = veiliger, minder gebruiksvriendelijk.

TOTP

Gedeelde sleutel + tijd → eenmalige codes.

Passkeys

WebAuthn: asymmetrische crypto, zonder wachtwoorden.