EPFL

SPRING
SECURITY AND PRIVACY ENGINEERING LABORATORY

CS-523 Advanced topics on Privacy Enhancing Technologies

Privacy-Preserving Authorization

Mathilde Raynal
SPRING Lab
mathilde.raynal@epfl.ch

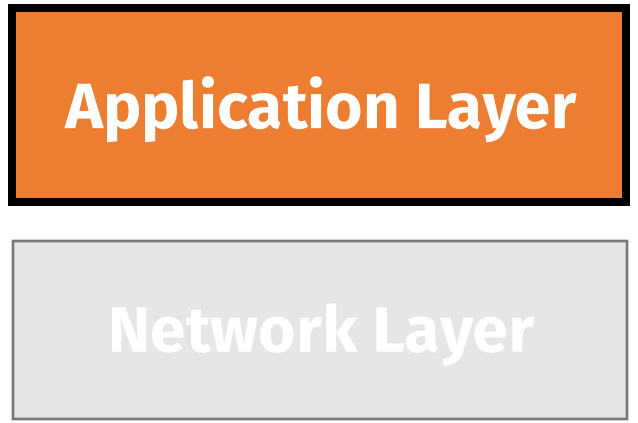
Slides credit: Carmela Troncoso, Wouter Lueks

Introduction
Privacy-Preserving Authorization

Course aim: learn **toolbox for privacy engineering**

tool
for building PETS

cryptographic
primitive



Goals

What should you learn today?

- Understand **when to use** privacy-preserving authorization

- Basic understanding of **zero-knowledge proofs**
  - Key properties
  - Schnorr example

- Understand what are **attribute-based credentials**:
  - Trust assumptions & key properties
  - How to choose attributes sets
  - Pointcheval-Sanders example

- Understand basic methods to implement **attribute-based credentials**
  - More in the "Secret Stroll" project!

- Understand **practical issues** when using anonymous authentication

Introduction

I can prove I am Mathilde

Content Mathilde is allowed to see

We know Mathilde and we know what she can access

I can prove I am Mathilde

Content Mathilde is allowed to see

Identity is not relevant!
Just that I am subscribed to certain content!

Authentication vs. authorization

Authentication

- Username and password
- Biometrics
- Client certificates
- Challenge response with public key cryptography (ssh)

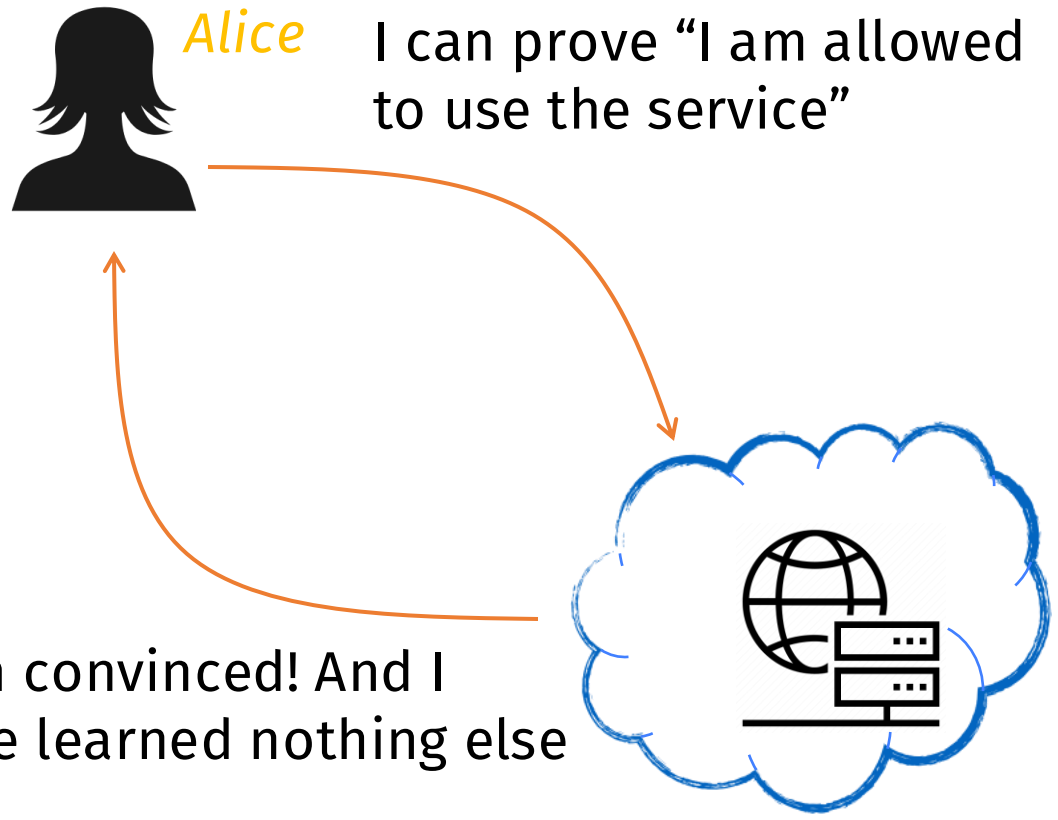
All of these *identify* the user. Is this always necessary?

Authorization

- Check that this user is a real person and not a bot (Cloudflare)
- is an honest editor (Wikipedia)
- paid for this service (video streaming, music, games, etc.)
- is old enough to access this service
- is allowed to vote

None of these require *identification*.
How do we build these?

Privacy-Preserving authorization



- A cryptographic primitive that enables users to prove possession of an attribute
- **Security property**: the proving party cannot lie, and the verifying party cannot be convinced if not true
- **Privacy property**: the verifying party cannot learn anything, other than the veracity of the statement proven (and what one infers from the statement itself)
- Privacy-preserving authorization builds on **zero-knowledge proofs**

Zero-Knowledge Proofs

Zero-knowledge proofs

A prover can use a **zero-knowledge proof** to prove that a **statement** is true, without revealing information beyond the fact that the statement is true.

Example 1 (In voting protocols)

Prover: Voter
Statement: This ciphertext *c* contains an encryption of 0 or 1.
Without revealing: the vote

Example 2 (PKI)

Prover: pk holder
Statement: They know the private key *sk* corresponding to this public key *pk.*
Without revealing: the private key *sk*

Zero-knowledge proofs: properties

A prover can use a **zero-knowledge proof** to prove that a **statement** is true, without revealing information beyond the fact that the statement is true.

Completeness: If the statement is true, an honest prover can convince an honest verifier that the statement is true.

Soundness: If the statement is false, a cheating prover cannot convince an honest verifier with high probability (i.e., close to 1).

Zero-knowledge: If the statement is true, no verifier learns anything other than the fact that the statement is true.

Deep Dive: Example 2 (PKI)

Prover: pk holder

Statement: They know the private key *sk* corresponding to this public key *pk.*

Without revealing: the private key *sk*

A solution: **Schnorr's proof of identification !**

Why does it have completeness? Soundness? ZK?

Schnorr's proof of identification

Peggy wants to prove to Victor that she knows the private key x corresponding to the public key h = g^x without revealing x.

Cyclic group: G
Generator: g
Group order: p (prime)
x ∈ $\mathbb{Z}_p$

Discrete logarithm problem:
Given g, h find x st. $h = g^x$

Peggy

Victor

Schnorr's proof of identification

Peggy wants to prove to Victor that she knows the private key x corresponding to the public key h = g^x without revealing x.

Cyclic group: G
Generator: g
Group order: p (prime)
x ∈ $\mathbb{Z}_p$

Discrete logarithm problem:
Given g, h find x st. $h = g^x$

Common information:

- Group: (G, g, p)
- Public key: $h = g^x$

Peggy

Victor

Schnorr's proof of identification (no c)

Peggy wants to prove to Victor that she knows the private key x corresponding to the public key h = g^x without revealing x.

Cyclic group: G
Generator: g
Group order: p (prime)
x ∈ $\mathbb{Z}_p$

Discrete logarithm problem:
Given g, h find x st. $h = g^x$

Common information:
- Group: (G, g, p)
- Public key: $h = g^x$

Peggy

Pick $r \in_R \mathbb{Z}_p$

$R = g^r$

$s = r + x \mod p$

Victor

$Rh =? g^s$

No: R = g^s / h always verifies

Completeness: If Peggy is honest, she can convince Victor (honest verifier) that the statement is true.

Soundness: If Peggy is not honest, she cannot convince Victor (honest verifier) with high probability.

Zero-knowledge: Victor cannot learn anything about x.

Schnorr's proof of identification (1-bit)

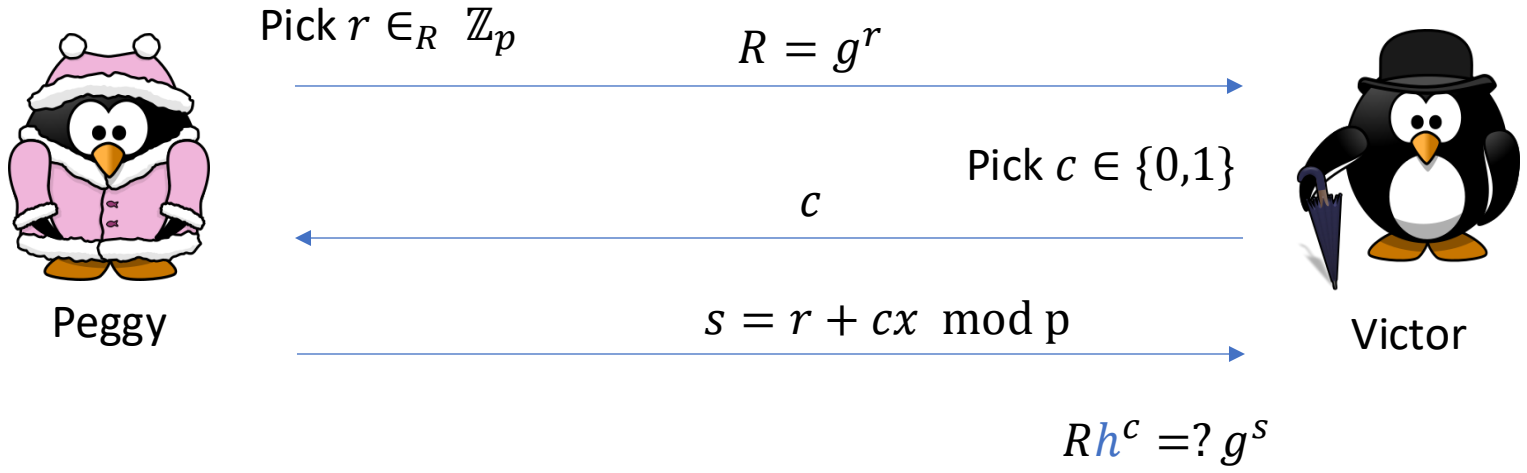
Peggy wants to prove to Victor that she knows the private key x corresponding to the public key h = g^x without revealing x.

Cyclic group: G
Generator: g
Group order: p (prime)
x ∈ $\mathbb{Z}_p$

Discrete logarithm problem:
Given g, h find x st. $h = g^x$

Common information:

- Group: (G, g, p)
- Public key: $h = g^x$



Schnorr's proof of identification (1-bit)

Peggy wants to prove to Victor that she knows the private key x corresponding to the public key h = g^x without revealing x.

Cyclic group: G
Generator: g
Group order: p (prime)
x ∈ $\mathbb{Z}_p$

Discrete logarithm problem:
Given g, h find x st. $h = g^x$

Common information:
- Group: (G, g, p)
- Public key: $h = g^x$

Peggy

Pick $r \in_R \mathbb{Z}_p$

$R = g^r$

Pick $c \in \{0,1\}$

c

$s = r + cx \mod p$

Victor

$Rh^c =? g^s$

Can guess c and use trick or nothing: 0.5

Completeness: If Peggy is honest, she can convince Victor (honest verifier) that the statement is true.

Soundness: If Peggy is not honest, she cannot convince Victor (honest verifier) with high probability.

Zero-knowledge: Victor cannot learn anything about x.

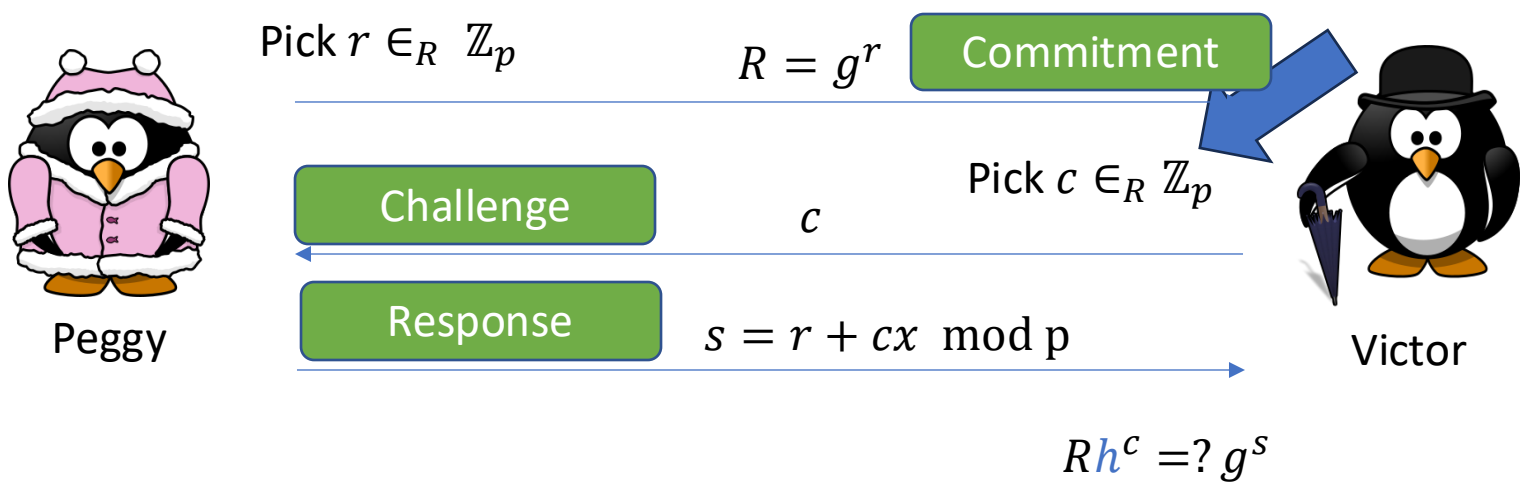
Schnorr's proof of identification (n-bits)

Peggy wants to prove to Victor that she knows the private key x corresponding to the public key h = g^x without revealing x.

Cyclic group: G
Generator: g
Group order: p (prime)
x ∈ $\mathbb{Z}_p$

Discrete logarithm problem:
Given g, h find x st. $h = g^x$

Common information:
- Group: (G, g, p)
- Public key: $h = g^x$



Guess with 1/p

Completeness: If Peggy is honest, she can convince Victor (honest verifier) that the statement is true.

Soundness: If Peggy is not honest, she cannot convince Victor (honest verifier) with high probability.

Zero-knowledge: Victor cannot learn anything about x.

Useful for project

Schnorr's proof of identification

Peggy wants to prove to Victor that she knows the private key x corresponding to the public key h = g^x without revealing x.

Cyclic group: G
Generator: g
Group order: p (prime)
x ∈ $\mathbb{Z}_p$

Discrete logarithm problem:
Given g, h find x st. $h = g^x$

Peggy

Short Notation:

$$\mathrm{PK}\{(x): h = g^x\}$$

Proving knowledge of a Pedersen's commitment:

$$\mathrm{PK}\{(x, r): C = g_1^x g_2^r\}$$

Victor

Completeness: If Peggy is honest, she can convince Victor (honest verifier) that the statement is true.

Soundness: If Peggy is not honest, she cannot convince Victor (honest verifier) with high probability.

Zero-knowledge: Victor cannot learn anything about x.

Non-interactive proofs: Fiat--Shamir heuristic

- **Interaction is costly,** requires communication rounds with verifier. Verifier needs to be **online.**

Fiat-Shamir heuristic:

- Turns interactive commitment-challenge-response protocols (called sigma-protocols) into non-interactive protocols
- Replace challenge *c* with cryptographic hash of:
  - All public values

  and
  - All commitments of the first step

Schnorr's proof of identification

Peggy wants to prove to Victor that she knows the private key x corresponding to the public key h = g^x without revealing x.

Cyclic group: G
Generator: g
Group order: p (prime)
x ∈ $\mathbb{Z}_p$

Discrete logarithm problem:
Given g, h find x st. $h = g^x$

Common information:

- Group: (G, g, p)
- Public key: $h = g^x$

Peggy

Pick $r \in_R \mathbb{Z}_p$

$R = g^r$

$c = H(g \parallel h \parallel R)$

R, c, $s = r + cx \mod p$

Victor

$Rh^c =? g^s$

Schnorr's proof of identification

Peggy wants to prove to Victor that she knows the private key x corresponding to the public key h = g^x without revealing x.

Cyclic group: G
Generator: g
Group order: p (prime)
x ∈ $\mathbb{Z}_p$

Discrete logarithm problem:
Given g, h find x st. $h = g^x$

Common information:

- Group: (G, g, p)
- Public key: $h = g^x$

Peggy

Pick $r \in_R \mathbb{Z}_p$

$R = g^r$

$c = H(g \parallel h \parallel R)$

~~R~~, c, $s = r + cx \mod p$

Victor

$R' = g^s h^{-c}$

$c' = H(g \parallel h \parallel R')$

$c =? c'$

~~$Rh^c =? g^s$~~

Schnorr's **signature**

Cyclic group: G
Generator: g
Group order: p (prime)
x ∈ $\mathbb{Z}_p$

Discrete logarithm problem:
Given g, h find x st. $h = g^x$

Peggy with public key h wants to sign m.

Common information:

- Group: (G, g, p)
- Public key: $h = g^x$
- Message m

Peggy

Pick $r \in_R \mathbb{Z}_p$

$R = g^r$

$c = H(g \parallel h \parallel R \parallel m)$

Signature c, $s = r + cx \mod p$

Victor

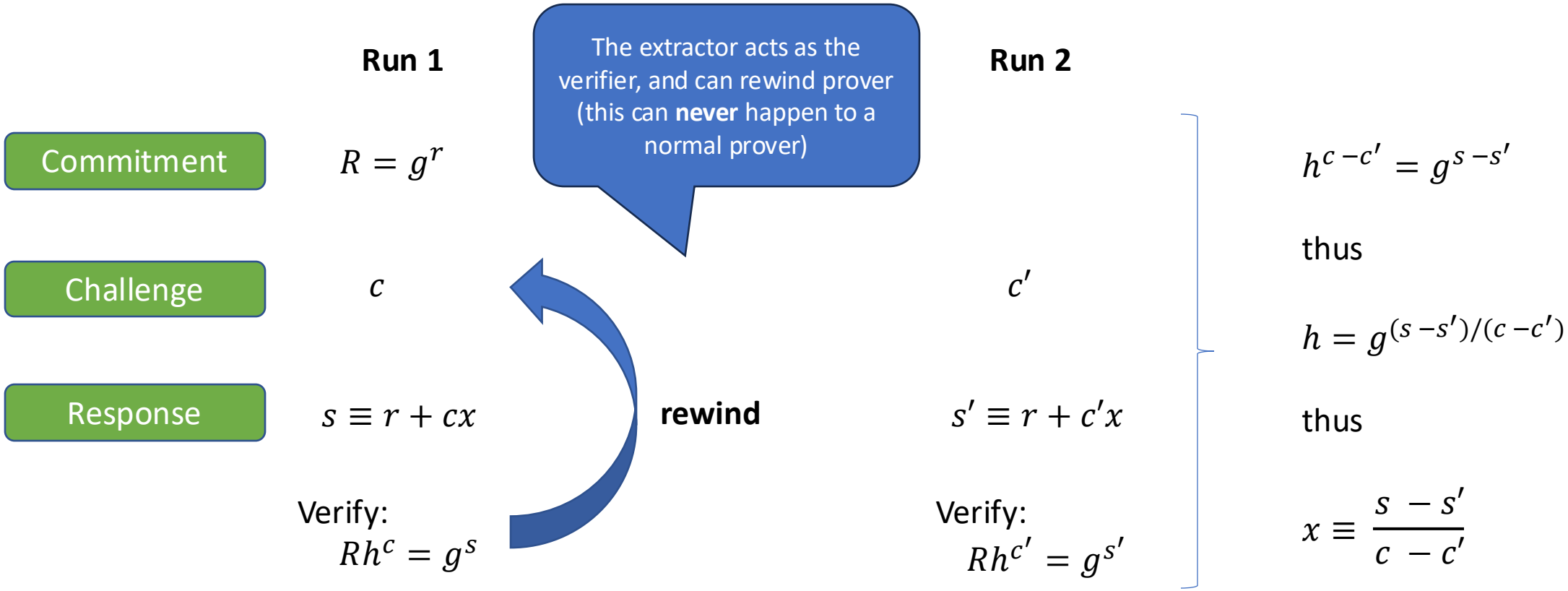
$R' = g^s h^{-c}$

$c' = H(g \parallel h \parallel R' \parallel m)$

$c =? c'$

Proof of Knowledge - Extractor

Proof of knowledge: There exists an extractor that, given a successful prover, can extract the witness (value of which knowledge is being proved).



Zero-knowledge proofs in the wild

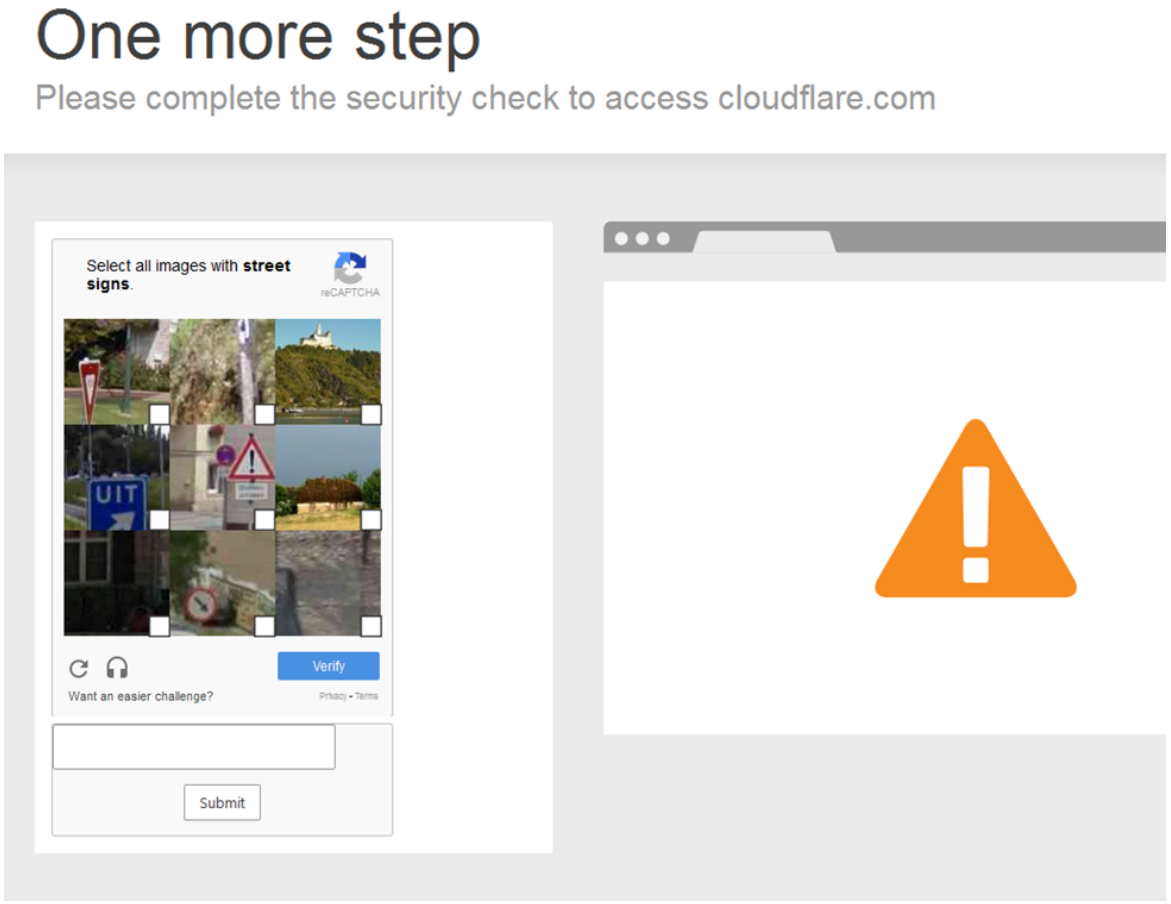
Critical building block in many cryptographic and privacy-enhancing technologies.

- Zcash digital currency (but uses a different type of proofs)
- Other types of electronic cash based on tokens
- Electronic voting systems
- Private (smart) metering
- Privacy friendly reputation system

Example: privacy pass

When browsing websites using Tor, users frequently have to solve CAPTCHAs.

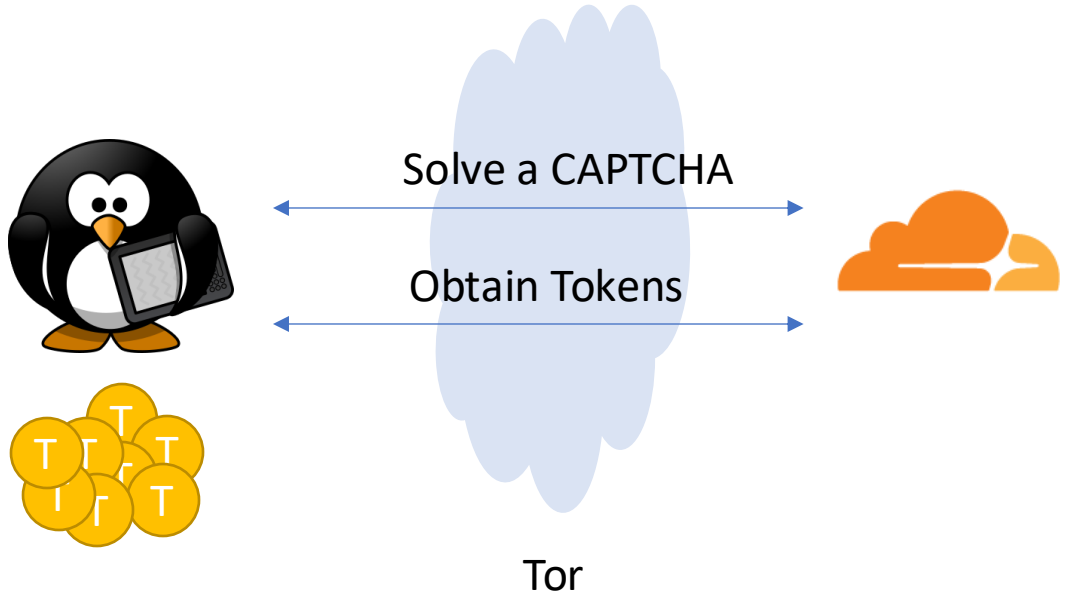
Why?



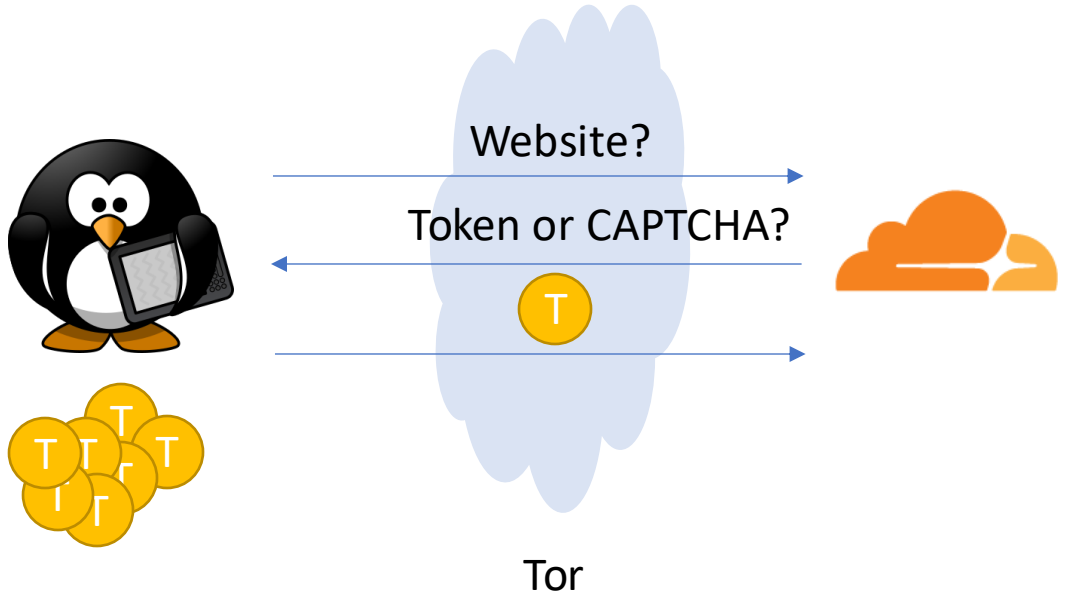
Example: privacy pass

- Goal: automatically allow real users (humans) from bots, implemented by Cloudflare and Tor browser.

Step 1: Obtain tokens



Step 2: Spend tokens instead of solving CAPTCHA

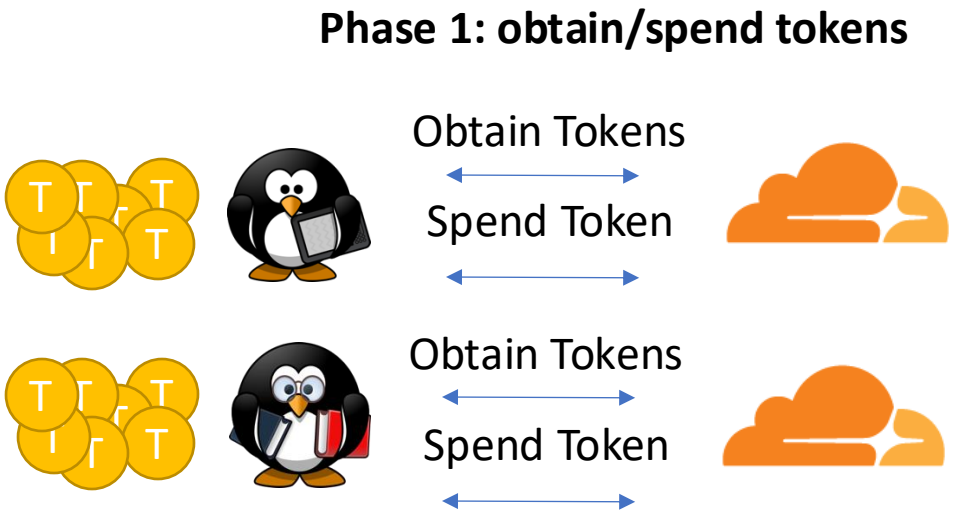


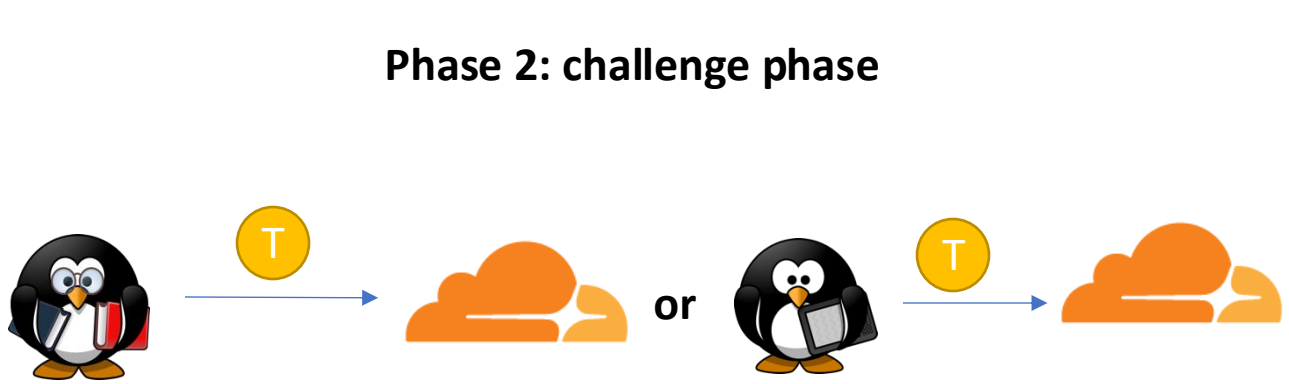
- What do we do ***not*** want?

  Cloudflare learning pairs: (“I am human Mathilde”, website I visit)

Property: unlinkability

- Cloudflare (the adversary) should not be able to link tokens by the same user.
- Modelled using an indistinguishability game which captures something stronger: can Cloudflare distinguish between two users?





Implementing privacy pass

Obtaining a token

Pick $t \in_R \{0,1\}^{128}$

Private key: x
Public key: $h = g^x$

t, T

Server has: private key *x*
Client has: value *t*
Client learns: T $= f(t, x)$ (but not *x*)
Server learns: nothing

OPRF Protocol

Cryptography sidebar

Cyclic group: G
Generator: g
Group order: p (prime)

Discrete logarithm problem:
Given g, h find x st. $h = g^x$

Hash function: $H: \{0,1\}^* \rightarrow G$

Showing a token

t, T

Check:
$f(t, x) = T$

Implementing privacy pass

Obtaining a token

Private key: x
Public key: $h = g^x$

Pick $t \in_R \{0,1\}^{128}$

Server has: private key *x*
Client has: value *t*
Client learns: T $= H(t)^x$ (but not *x*)
Server learns: nothing

t, T

Showing a token

t, T

Check:
$f(t, x) = T$

Cryptography sidebar

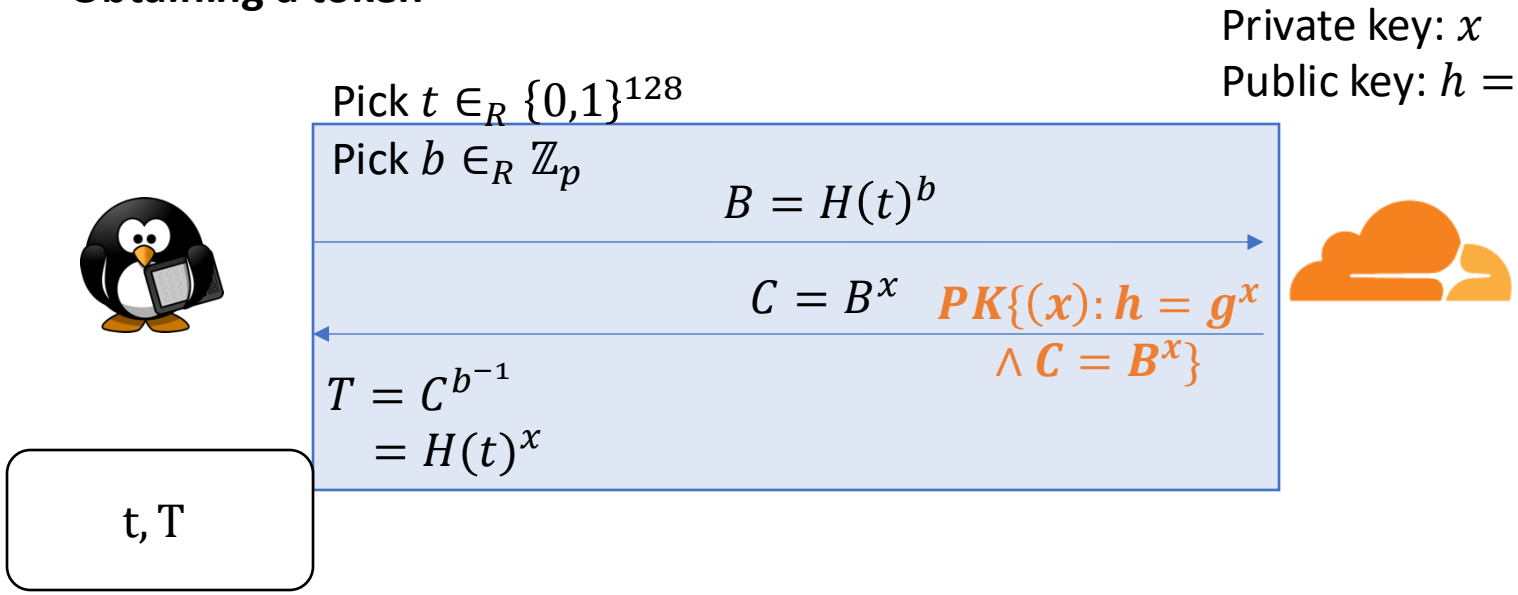
Cyclic group: G
Generator: g
Group order: p (prime)

Discrete logarithm problem:
Given g, h find x st. $h = g^x$

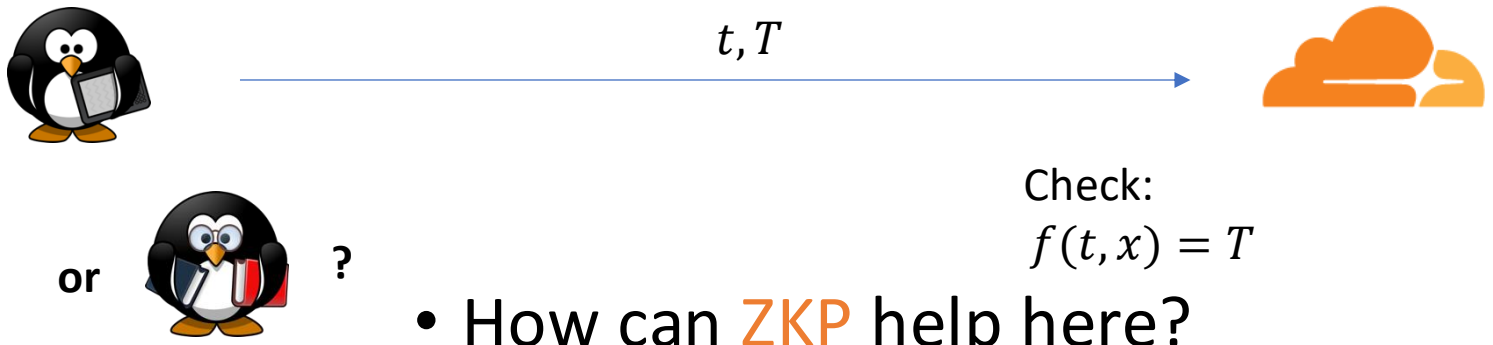
Hash function: $H: \{0,1\}^* \rightarrow G$

Implementing privacy pass

Obtaining a token



Showing a token



- How can ZKP help here?

Cryptography sidebar

Cyclic group: G
Generator: g
Group order: p (prime)

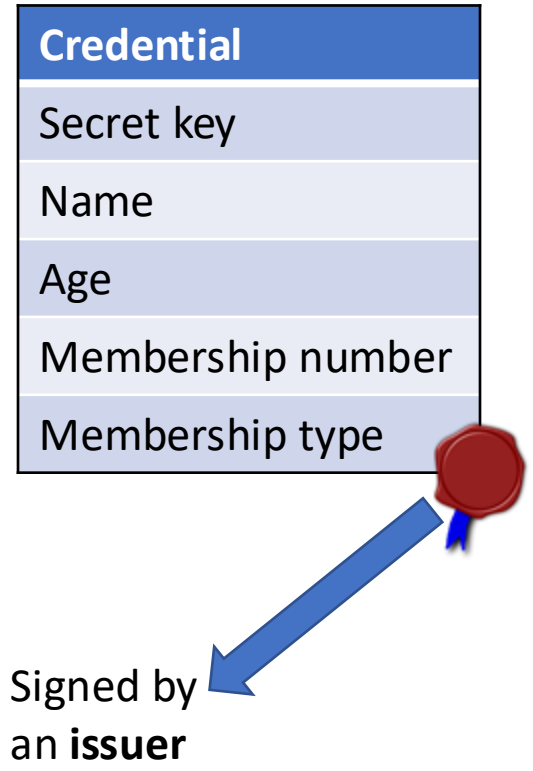
Discrete logarithm problem:
Given g, h find x st. $h = g^x$

Hash function: $H: \{0,1\}^* \rightarrow G$

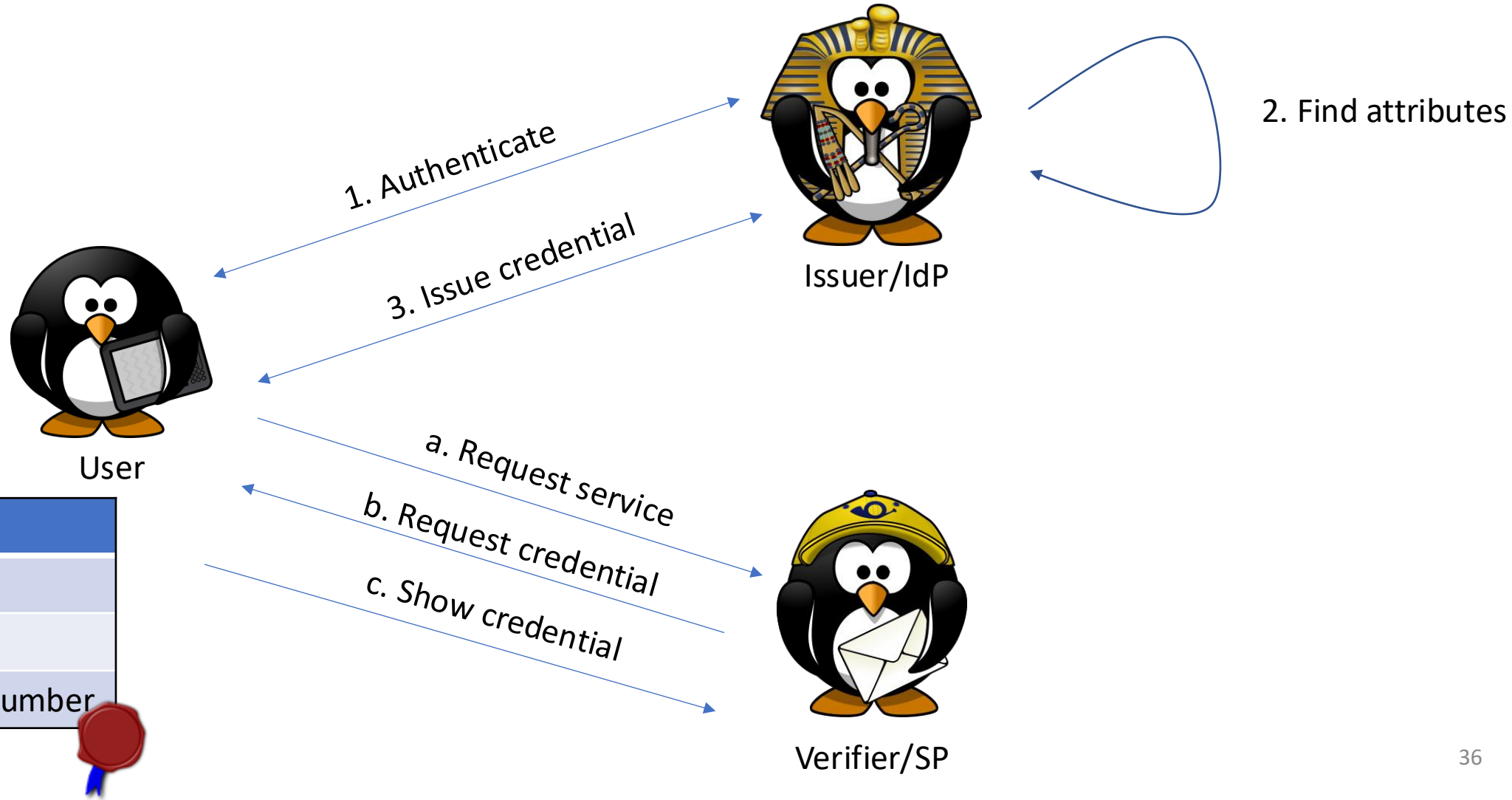
Attribute-based credentials

Attribute-based credentials

- Also known as *anonymous credentials*
- As opposed to tokens, can contain other attributes
- Attributes are encoded as numbers, may represent:
  - Membership status (normal user, premium user)
  - Name
  - Age
  - Social security number
  - Random identifiers and secret keys
  - Application specific identifiers
  - …



Obtaining credentials and showing credentials



Attribute-based credentials: properties

Unforgeability: only the issuer should be able to produce valid credentials.

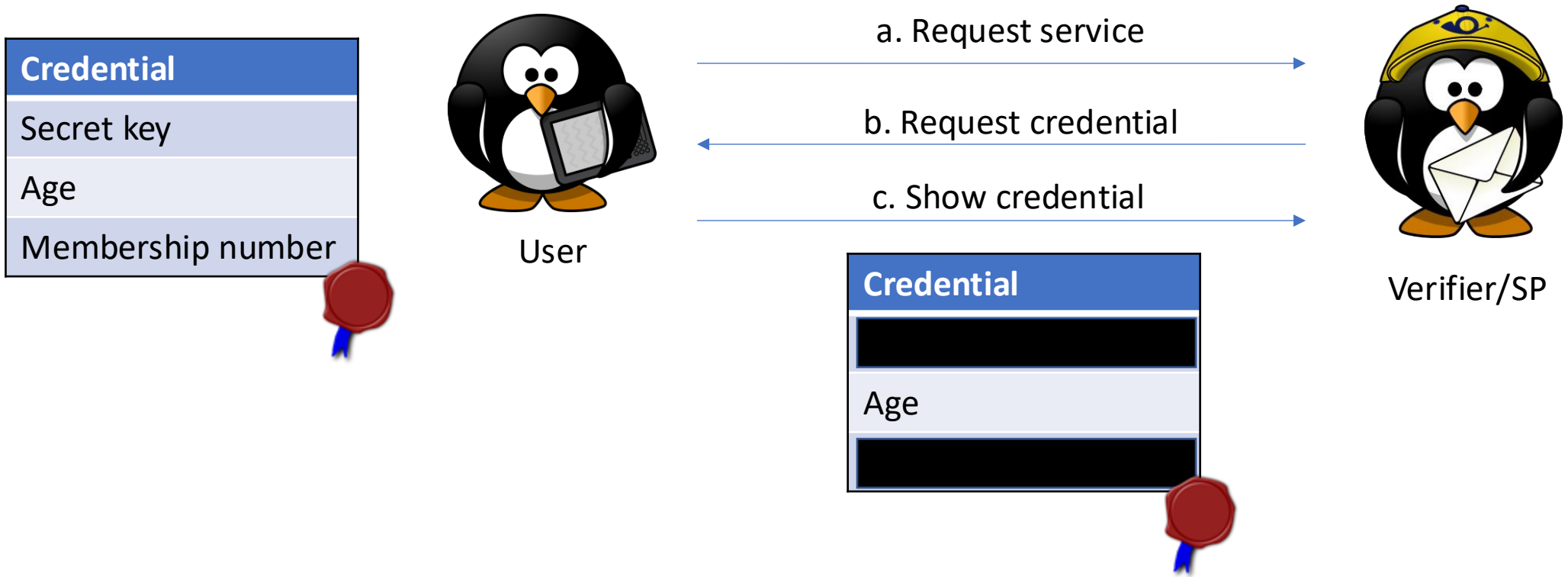
Selective disclosure: the user can hide irrelevant attributes

Issuer unlinkability: the issuer should not be able to recognize a credential that it previously issued

Verifier unlinkability: the verifier should not be able to link two consecutive showings of the same credential

Selective disclosure

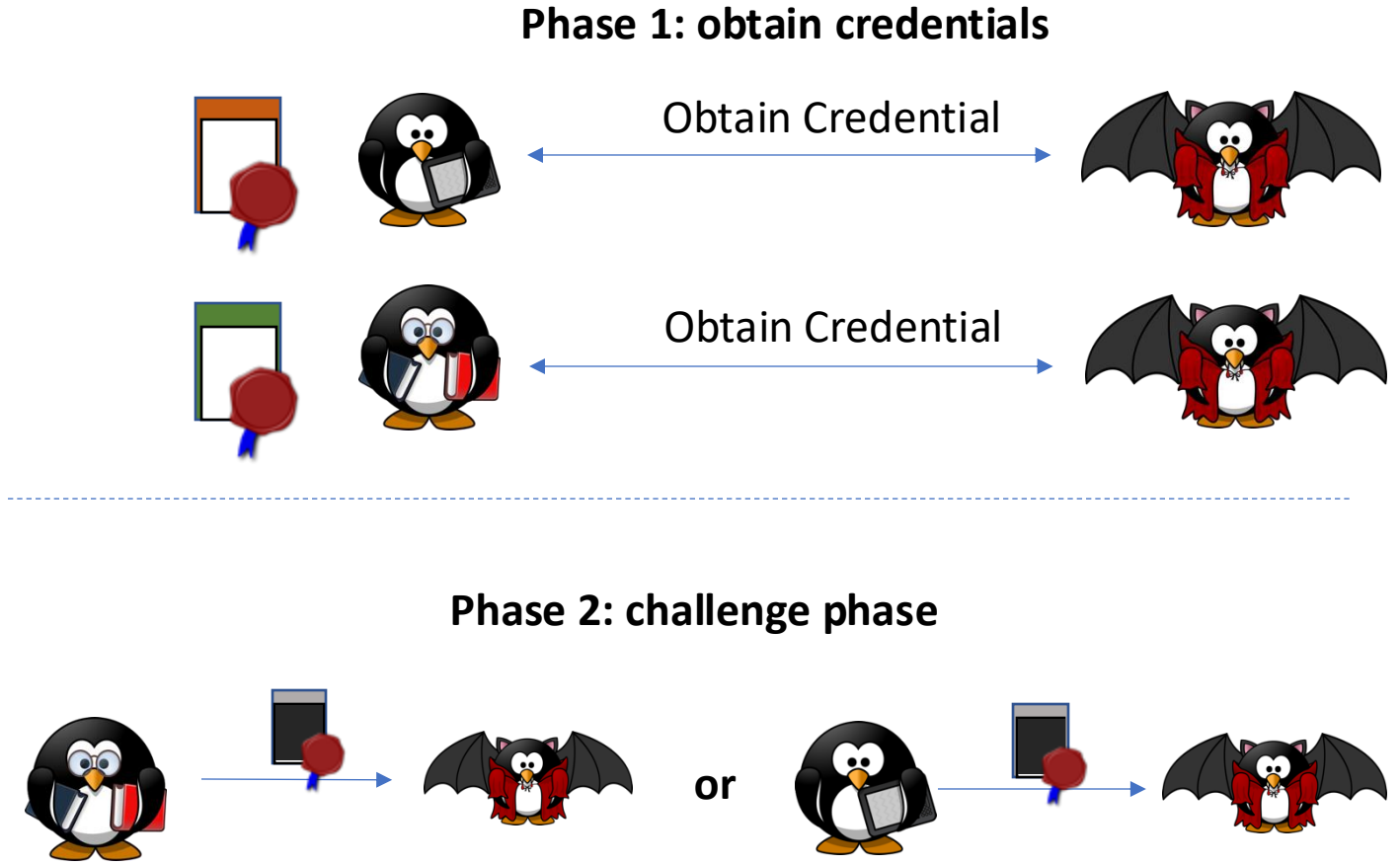
The user can hide irrelevant attributes. But the verifier can still check the validity of the credential.



Construction 1: traditional signatures

Issuer Unlinkability

- The issuer should not be able to recognize a credential that it previously issued
- Modelled using an indistinguishability game.
- Phase 1: obtain credentials
- Phase 2: challenge phase, distinguish users



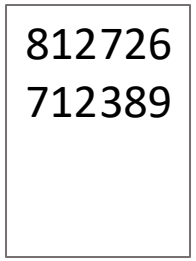
Note: both credentials should “look” the same, they should disclose the same attributes.

Construction 2: blind signatures

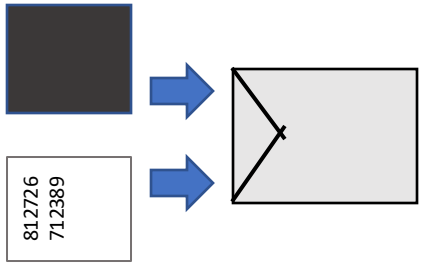
- Blind signature: signer signs a message *m* without knowing what it signs. Moreover, it cannot later recognize this signature.
- Property: exactly as issuer unlinkability.
- Implemented by: U-Prove by Microsoft (2000), but also Anonymous Credentials Light (2013), and PrivacyPass

A simple physical blind signature scheme

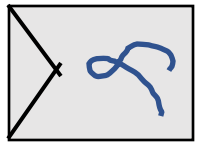
Step 1: write down serial number



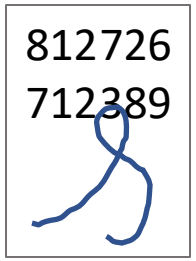
Step 2: place in envelope with carbon paper



Step 3: issuer signs the envelope



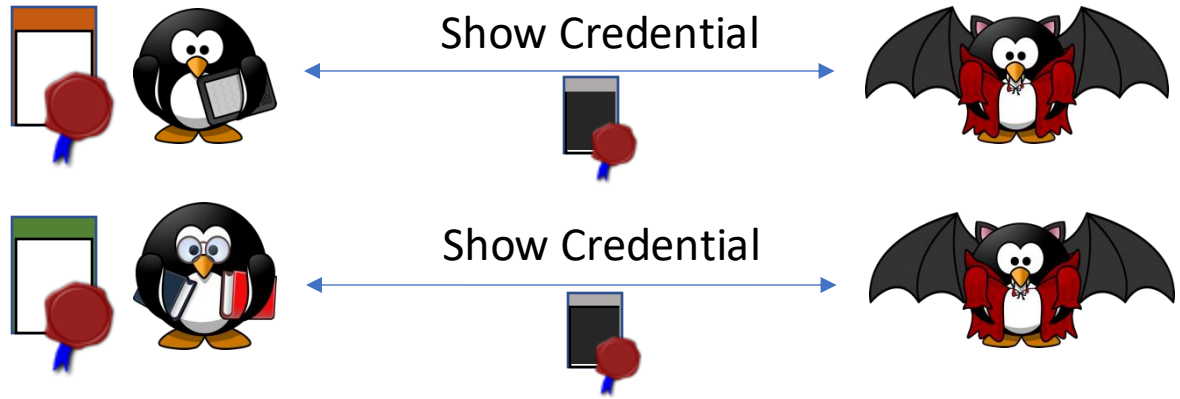
Step 4: user recovers signed statement



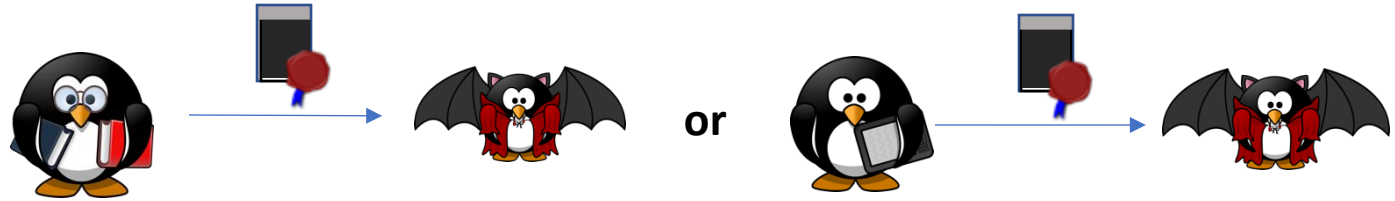
Verifier (multi-show) Unlinkability

- The verifier should not be able to recognize a credential that it previously saw
- Modelled using an indistinguishability game.
- Phase 1: see credentials for different users
- Phase 2: challenge phase, distinguish users

Phase 1: see credentials



Phase 2: challenge phase



Note: both credentials should "look" the same, they should disclose the same attributes (key and value).

Construction 3: proving having signature

Issuing a credential

1. Commit to user-defined attributes

3. Return signature σ on attributes

Not always a blind signature

2. Find other attributes

Credential
Secret key
Age
Membership number

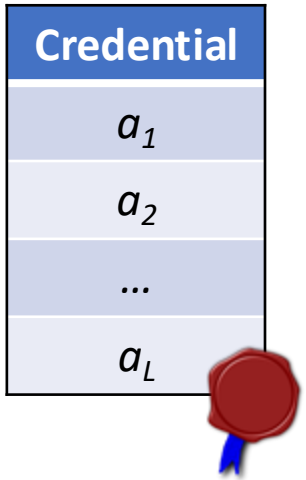
Showing a credential

User

User proves: “I have a valid signature σ on some attributes”

Verifier/SP

Deep Dive: Pointcheval-Sanders credential



A solution: **Pointcehval-Sanders**
Why does it have unforgeability? Selective disclosure? I&V unlinkability?

Pointcheval-Sanders credential

The next 3 slides are full of math, AGAIN 😔

but then we're done for today 🙂

Pointcheval-Sanders credential

Pointcheval-Sanders **signatures**

Key generation

- Pick generator $\tilde{g} \in_R G$
- Private key: $(x, y_1, \dots, y$
- Public key:
  $(\tilde{g}, \tilde{X}, \tilde{Y}_1, \dots, \tilde{Y}_L) = (\tilde{g}$

Cyclic group
Generators:
Group order: p

Pairing: $e : G_1 \times G_2 \to G_T$
Bilinear: $e(g^a, \tilde{g}^b) = e(g, \tilde{g})^{ab}$

Details in handout

Signing tuple $(m_1, \dots, m_L)$

- Pick $h \in_R G_1^*$ and output
- Signature $\sigma = \left(h, h^{x+\sum y_i m_i}\right)$

Verifying a signature:

- Given a signature $\sigma = (\sigma_1, \sigma_2)$
- Message: $(m_1, \dots, m_L)$
- And public key $(\tilde{g}, \tilde{X}, \tilde{Y}_1, \dots, \tilde{Y}_L)$
- Check:
  - $\sigma_1 \neq 1_{G_1}$
  - $e\left(\sigma_1, \tilde{X} \cdot \prod \tilde{Y}_i^{m_i}\right) = \mathrm{e}(\sigma_2, \tilde{g})$

Pointcheval-Sanders credential

Issuing a PS credential

- User commits to hidden attributes, pick $t \in_R \mathbb{Z}_p$:

$$C = g^t \prod_{i \in U} Y_i^{a_i}$$

- And proves that she did so correctly:

$$PK\left\{(t, (a_i)_{i \in U}) : C = g^t \prod_{i \in U} Y_i^{a_i}\right\}$$

- Issuer verifies the proof, picks $u \in_R \mathbb{Z}_p$ and sets:

$$\sigma' = \left(g^u, \left(XC \prod_{i \in I} Y_i^{a_i}\right)^u\right)$$

- User forms signature $\sigma = (\sigma_1', \frac{\sigma_2'}{\sigma_1'^t})$

Proof shows that C is correctly formed. What goes wrong if you omit this?

Attribute sets:
U: attributes **determined by user** (hidden from issuer)
I: attributes **determined by issuer**

Cyclic groups: G_1, G_2, G_T
Generators: $g, \tilde{g}, g_T$
Group order: p

Pairing: $e : G_1 \times G_2 \to G_T$
Bilinear: $e(g^a, \tilde{g}^b) = e(g, \tilde{g})^{ab}$

PS Signatures
Private key:
$(x, y_1, \ldots, y_L) \in_R \mathbb{Z}_p$
$X = g^x$
Public key:
$(g, Y_i) = (g, g^{y_i}) \in G_1$
$(\tilde{g}, \tilde{X}, \tilde{Y}_i) = (\tilde{g}, \tilde{g}^x, \tilde{g}^{y_i}) \in G_2$

Signature: $\sigma = (\sigma_1, \sigma_2)$ such that
$\sigma = (h, h^{x + \Sigma y_i a_i}) \in G_1^2$

Pointcheval-Sanders credential

Proving **possession** of a credential

- User has a credential $\sigma = (\sigma_1, \sigma_2)$ on $(a_1, \dots, a_L)$
- Pick $r, t \in_R \mathbb{Z}_p$ and compute $\sigma' = (\sigma_1^r, (\sigma_2 \sigma_1^t)^r)$
- Send σ' and disclosed attributes $(a_i)_{i \in D}$
- And proves that the signature is valid:

$$PK\left\{(t, (a_i)_{i \in H}) : \frac{e(\sigma_2', \tilde{g}) \prod_{i \in D} e(\sigma_1', \tilde{Y}_j)^{-a_i}}{e(\sigma_1', \tilde{X})} = e(\sigma_1', \tilde{g})^t \prod_{i \in H} e(\sigma_1', \tilde{Y}_i)^{a_i}\right\}$$

- The verifier accepts if proof is valid and $\sigma_1' \neq 1$

What goes wrong if you omit this?

Attribute sets:
H: hidden attributes
D: disclosed attributes
Different from previous slide!!!

Cyclic groups: G_1, G_2, G_T
Generators: $g, \tilde{g}, g_T$
Group order: p

Pairing: $e : G_1 \times G_2 \to G_T$
Bilinear: $e(g^a, \tilde{g}^b) = e(g, \tilde{g})^{ab}$

PS Signatures
Private key:
$(x, y_1, \dots, y_L) \in_R \mathbb{Z}_p$
$X = g^x$
Public key:
$(g, Y_i) = (g, g^{y_i}) \in G_1$
$(\tilde{g}, \tilde{X}, \tilde{Y}_i) = (\tilde{g}, \tilde{g}^x, \tilde{g}^{y_i}) \in G_2$

Signature: $\sigma = (\sigma_1, \sigma_2)$ such that
$\sigma = (h, h^{x + \sum y_i a_i}) \in G_1^2$

$e(\sigma_1', \tilde{g}^t \tilde{X} \cdot \prod \tilde{Y}_i^{a_i}) = e(\sigma_2', \tilde{g})$

Pointcheval-Sanders credential

Properties

- **Unforgeability**: yes, from the PS *signatures*
- **Selective disclosure**: yes, use proof of knowledge to prove that the signature is valid without revealing all the attributes
- **Issuer & verifier unlinkability:** yes. Informally, the randomization and the proof of knowledge hide the signature. Therefore, neither the issuer (signer) nor the verifier can recognize it.

ABCs in the wild

- Algebraic MACs (2014): assumes issuer and verifier are same, but has multi-show unlinkability, does not require pairings
- Anonymous Credentials Light (2013): uses the blind-signature paradigm, only single show, does not require pairings
- IRMA (Irma.app) implements Idemix on a smart phone app + provides support for Identity Providers and Service Providers

Other credential schemes

- U-Prove by Stefan Brands/Microsoft around 2000; single show
  - Based on the blind signing paradigm
  - Standard discrete-logarithm based setting, no pairings
  - To get unlinkability: use a credential only once, no verifier unlinkability
- Identity Mixer (Idemix) by Jan Camenisch and Anna Lysyanskaya/IBM research around 2002; multi-show
  - Based on signature scheme + proof of knowledge of signature
  - Setting 1: strong RSA assumption, large key sizes required
  - Setting 2: elliptic curve/pairing based setting
  - Supports a large number of extensions, including range proofs, key escrow, domain specific pseudonyms

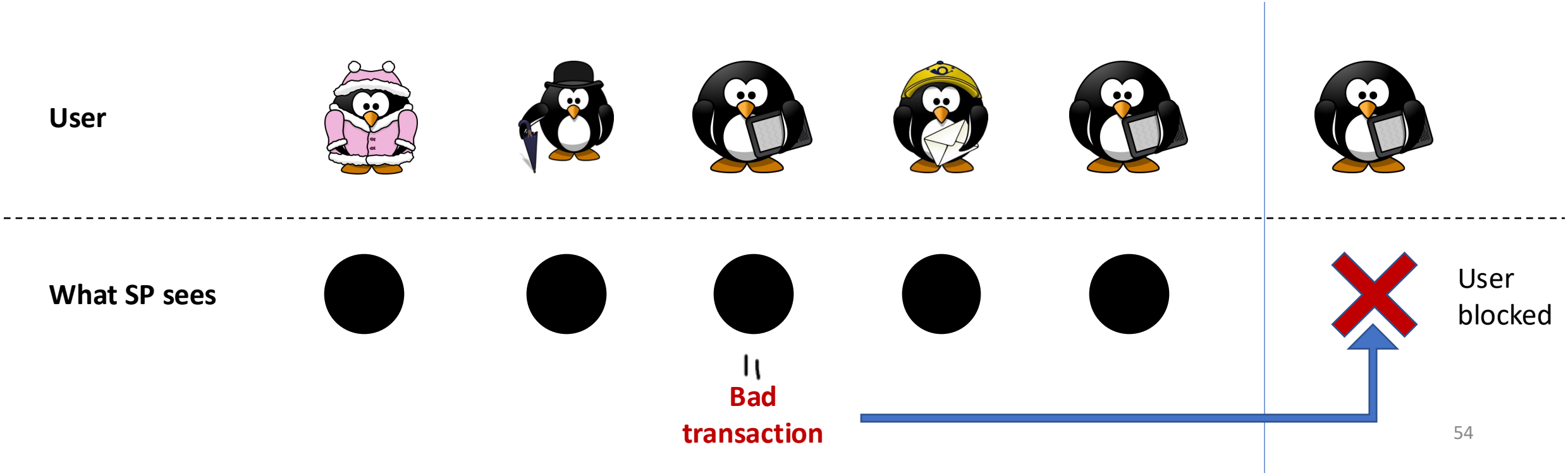
Example: Today’s Live Exercises

Revoking/blocking a credential

- The revoke/block a credential means to invalidate it
- Reasons:
  - User detects credentials are stolen
  - Issuer decides to withdraw statements
  - Credentials are being abused
- Questions to ask:
  - Who can initiate revocation/blocking? What information is needed?
  - Can users detect that they have been revoked/blocked? (Or can the revocation test be made in silence?)
  - Does the system provide backward unlinkability after revocation?

Blocklistable anonymous credentials

- What if we do *not* know the user's identity?
- -> block misbehaving anonymous users without needing to identify them

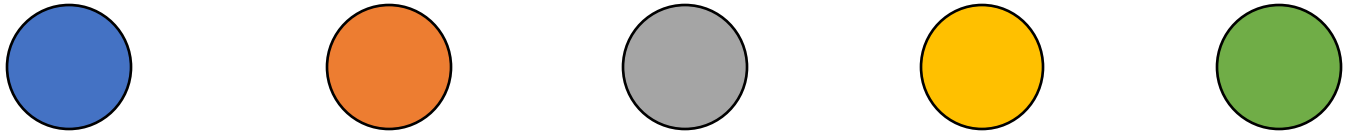


Blocklistable anonymous credentials, idea

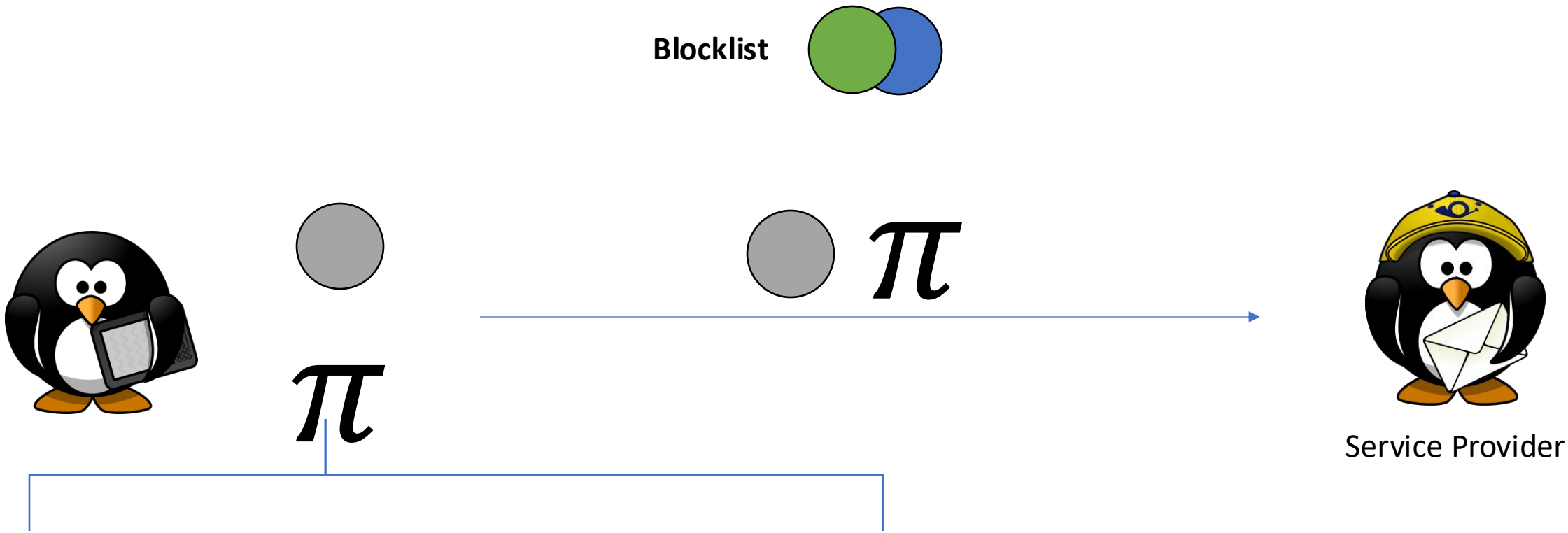
- For every transaction, user produces a token
- Tokens belong to users, but SP cannot determine which user
- Users use a credential to prove that the token is correctly formed

User

Token for SP



Blocklistable anonymous credentials, idea II



- Token ● is correct given credential
- None of the tokens on the blocklist are mine

Constructing tokens and proofs

- Every user has a credential σ, on secret s
- Tokens: $(h, t = h^s)$ where h a random element
- Blocklist: $BL = \{(h_1, t_1), \dots, (h_n, t_n)\}$
- Construct the proof:

$$PK\left\{(\sigma): \begin{array}{c} \sigma \text{ over } s \ \wedge \\ t = h^s \ \wedge \\ \bigwedge_{i=1}^{n} (t_i \neq h_i^s) \end{array}\right\}$$

Insert: big equation from before

Have credential

Token correct

Blocklisted tokens not mine

User driven revocation

- Tokens: $(h, t = h^s)$ where h a random element (again, prove correctness of this tuple w.r.t. user's credential)
- To revoke a credential, user makes *s* public
- Now verifiers can check a token (h, t) against all revoked $s_1, \ldots, s_n$

Challenges:

- Check is linear in size of the revocation list (for verifier, constant time for user)
- Backward linking is possible, once secret *s* is known

Issuer driven revocation

- Issuer adds random attribute a_0
- User constructs token: $(h, t = h^{a_0})$ where h a random element (again, prove correctness of this tuple w.r.t. user's credential)
- Now issuer can reveal a_0 to revoke a credential

Challenges:

- Know who you want to block
- Issuer can trace users without their knowledge
- Does not give backward unlinkability

Alternative: use accumulators to hold a blocklist. Does not suffer from backward linkability.

Limiting the number of uses

n-times anonymous credentials can be used for at most *n* times. Thereafter, verifiers can recognize reuse.

How?

- Before: in blocklistable anonymous credentials, users can make unlimited number of tokens
- Idea: limit the number of tokens a user can create

Goals

What should you learn today?

- Understand **when to use** privacy-preserving authorization

- Basic understanding of **zero-knowledge proofs**
  - Key properties
  - Schnorr example

- Understand what are **attribute-based credentials**:
  - Trust assumptions & key properties
  - How to choose attributes sets
  - Pointcheval-Sanders example

- Understand basic methods to implement **attribute-based credentials**
  - More in the "Secret Stroll" project!

- Understand **practical issues** when using anonymous authentication

References

- Nigel Smart. Cryptography: An Introduction. 3rd edition online. Also available in print. Chapters 11 and 25.
- https://privacypass.github.io/protocol/
- David Pointcheval, Olivier Sanders. “Short Randomizable Signatures”. In CT-RSA 2016.
- Patrick P. Tsang, Man Ho Au, Apu Kapadia, Sean W. Smith. “BLAC: Revoking Repeatedly Misbehaving Anonymous Users without Relying on TTPs”. ACM Trans. Inf. Syst. Secur. 13(4) 2010.
- Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. “How to win the clonewars: efficient periodic *n*-times anonymous authentication”. In: CCS 2006.
- Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. “An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials”. In: PKC 2009.