COM - 402

Crypto-based Solutions

- Homomorphic encryption
- Attribute-based credentials

Adapted from a slide show provided by Wouter Lueks; the slides on homomorphic encryption were produced by Sylvain Chatel

Homomorphic Encryption

Terminology

Plaintext space

Given a cryptosystem, we denote $\mathcal{P}$ its plaintext space: the set of all possible messages m

Ciphertext space

Given a cryptosystem, we denote C its ciphertext space: the set of all possible ciphertexts

Group homomorphism

Given $(G,\boxplus)$, $(H,\boxtimes)$ two groups, the function $h: G \mapsto H$ is a group homomorphism if

$$\forall u, v \in G, \qquad h(u \boxplus v) = h(u) \boxtimes h(v)$$

Homomorphic Encryption

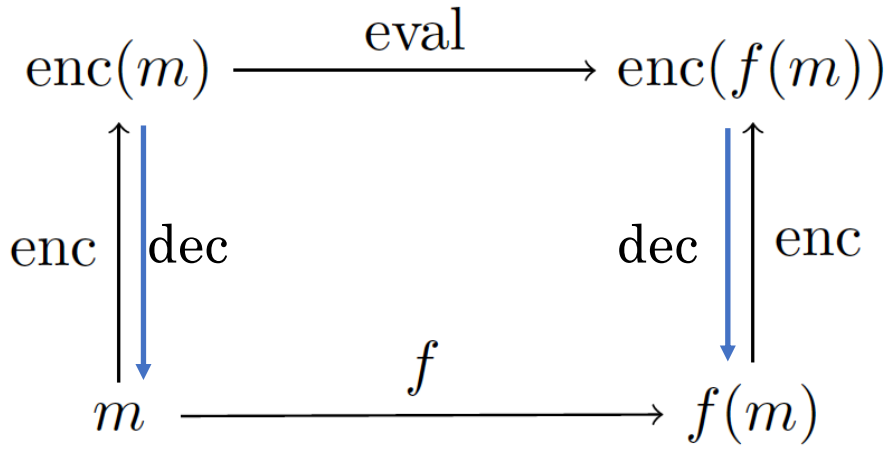
- Homomorphic cryptosystems are malleable by nature
- **There exists an homomorphism between their ciphertext and plaintext space**

  $\mathrm{h}(\mathrm{x}) = Dec(x)$ is an homomorphism between $\mathcal{C}$ and $\mathcal{P}$

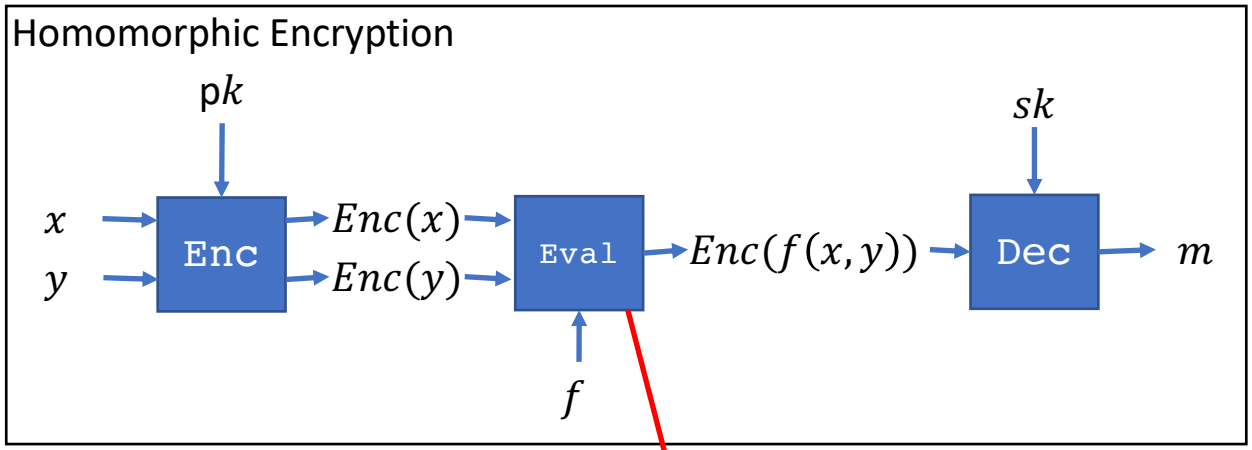
Some cryptosystems are **partially** homomorphic
→ they support **one** arithmetic operation

$Enc : (\mathcal{P}, \boxplus) \mapsto (\mathcal{C}, \boxtimes)$

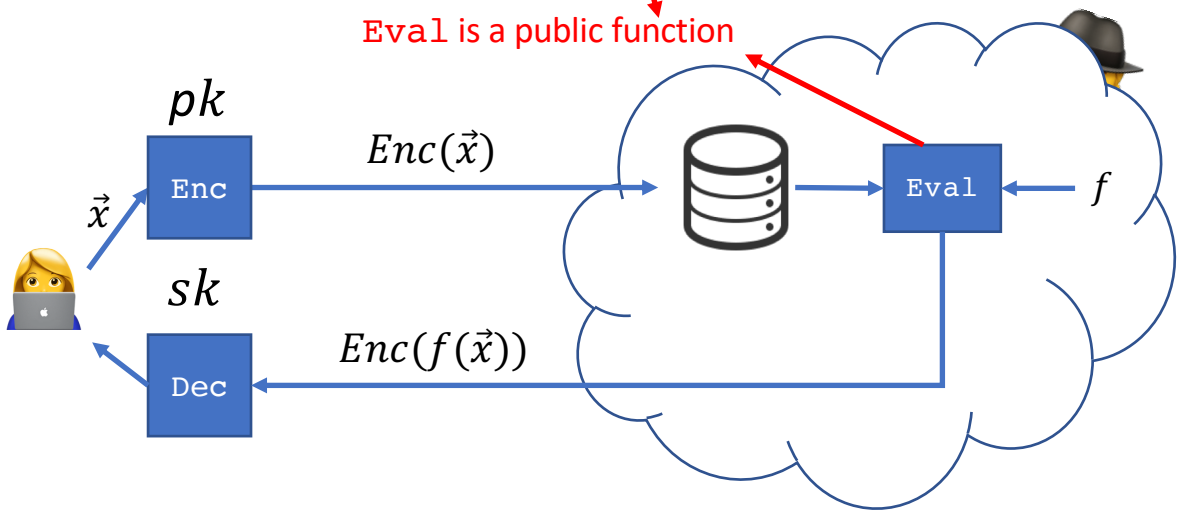
$\forall m_1, m_2 \in \mathcal{P}, Enc(m_1) \boxtimes Enc(m_2) = Enc(m_1 \boxplus m_2)$



Homomorphic encryption



In Processing



Homomorphic Encryption - Paillier

<u>Additively Homomorphic</u>: Paillier Cryptosystem

Let p, q be two independent primes st. $\gcd\big(pq, (p-1)(q-1)\big) = 1$.
We define $n = pq$ a RSA modulus and $\lambda = \varphi(n) = \varphi(p,q) = (p-1)(q-1)$.
Let $\mu = \varphi(n)^{-1} \, mod \, n$.
Let (λ, μ) be the private key and n the public key.

For message $m \in P$, and $r \leftarrow \mathbb{Z}_{n^2}^*$

$$\boldsymbol{Enc}(m) = (1+n)^m r^n \ mod \, n^2$$

$$\boldsymbol{Dec}(c) = \frac{\left[c^{\varphi(n)} \, mod \, n^2\right] - 1}{n} (\varphi(n))^{-1} \, mod \, n$$

There exists a group homomorphism Enc : $(P, +) \longrightarrow (C, \times)$:
For messages $m_1, m_2 \in P$, and $r_1, r_2 \leftarrow \mathbb{Z}_{n^2}^*$

$$Enc(m_1) \cdot Enc(m_2) \equiv (1+n)^{m_1} r_1^n (1+n)^{m_2} r_2^n \ \ mod \, n^2 \equiv (1+n)^{m_1+m_2} (r_1 r_2)^n \ \ mod \, n^2$$

$$\equiv Enc(m_1 + m_2) \ \ mod \, n^2$$

$'\leftarrow'$ means sampled from

Homomorphic Encryption - Paillier

Paillier Cryptosystem : Example

Let p = 11, q= 17.
Thus, $n = 11 * 17 = 187$ and $n^2 = 34969$.
$\varphi(n) = 160$.

$m = 175, r = 83 \in \mathbb{Z}_{187}^{*}$.

The ciphertext is $c = (1 + 187)^{175}.83^{187} mod\ 34969 = 23911$

The decryption is then :

$$\widehat{m} = \left[\frac{(23911^{160}\ mod\ 34969) - 1}{187}\right] .[160^{-1}\ mod\ 187\ ]\ mod\ 187$$

$$= \left[\frac{25620\ - 1}{187}\right] .90\ \ mod\ 187\ = 137\ .90\ mod\ 187 = 175 = m$$

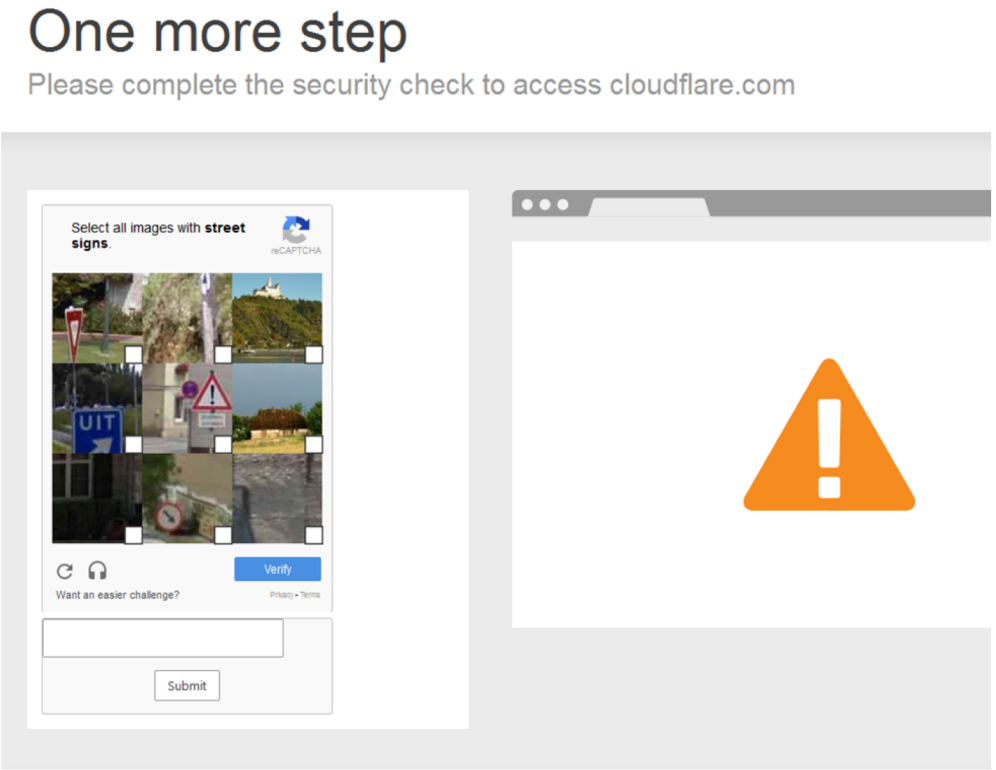
Example from J. Katz, Y.Lindell, "Introduction to Modern Cryptography: Principles and Protocols," Chapman & Hall/CRC, 2007 p.414

Attribute-Based Credentials

Browsing website using Tor

When browsing websites using Tor, users frequently have to solve captcha's?

Why?



Editing Wikipedia via Tor

Wikipedia does not allow:

- Editing pages when using the Tor Browser
- Create accounts when using the Tor Browser

Why?

How do you solve this normally

Authentication

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

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

What you really want: authorization

Check that this user

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

None of these require *identification*.

Passports: issuers and verifiers

Why do you trust the data in a passport?

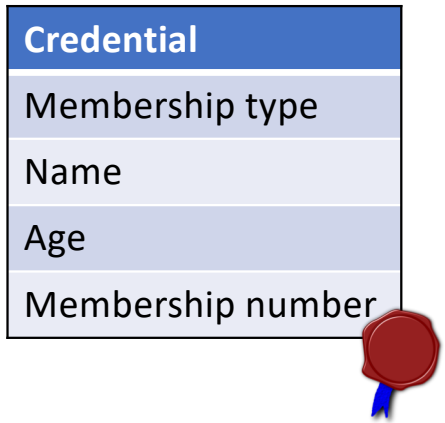
- Issued by a reliable country
- The passport looks authentic (i.e., it is not fake)

Issuer or Identity Provider (IdP): entity that validates the information in the documents

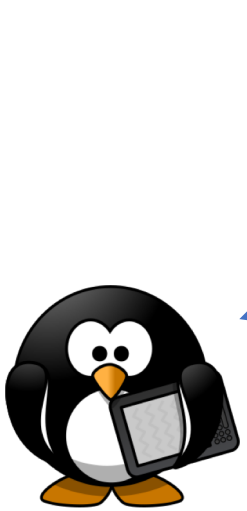
Verifier or Service Provider (SP): entity that verifies that the information is correct and provides a service

Attribute-based credentials

- Digital variant of passports, driver's license, etc.
- Also known as anonymous credentials
- As opposed to tokens, can contain other attributes
- Attributes are encoded as numbers, may represent, e.g.,
  - Membership status (normal user, premium user)
  - Name
  - Age
  - Social security number
  - Random identifiers
  - Application-specific identifiers



Obtaining credentials and showing credentials



User

Issuer/IdP

1. Authenticate

3. Issue credential

2. Find attributes

a. Request service

b. Request credential

c. Show credential

Verifier/SP

Credential
Secret key
Age
Membership number

What properties should these credentials satisfy?

Properties of attribute-based credentials

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

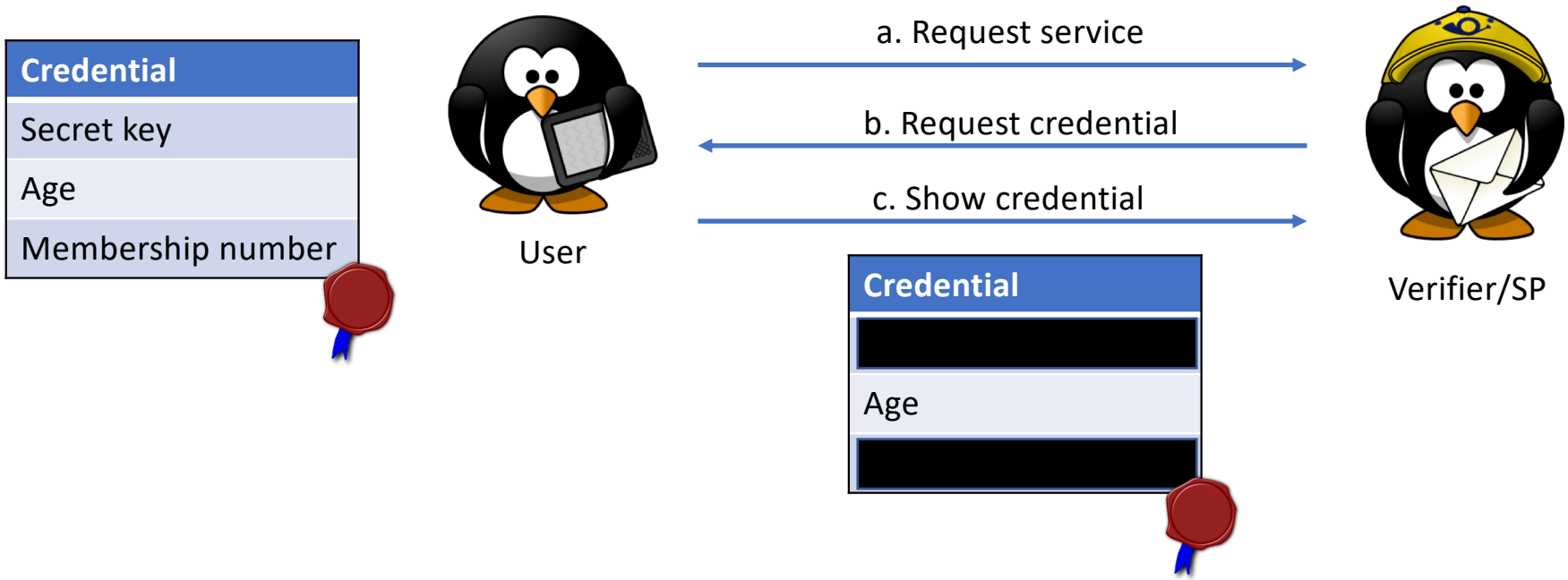
Selective disclosure: the user can hide irrelevant attributes

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

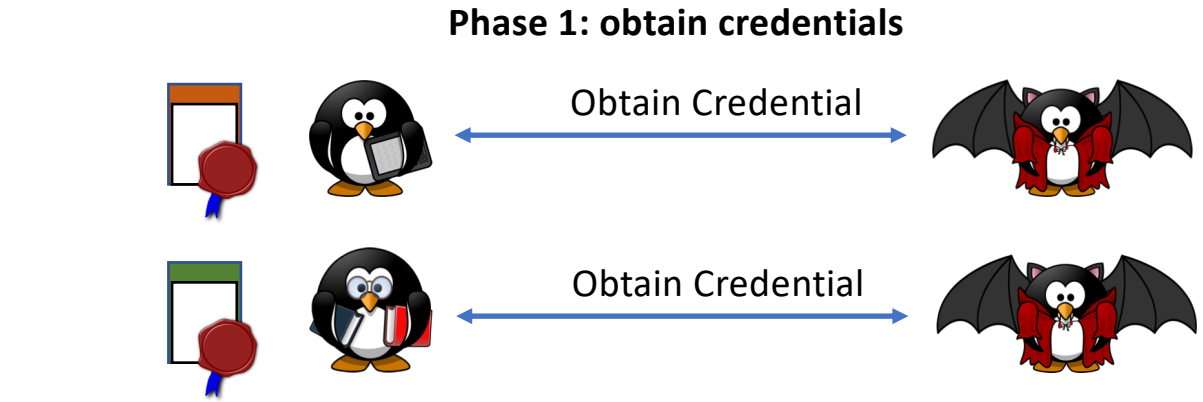
Selective disclosure

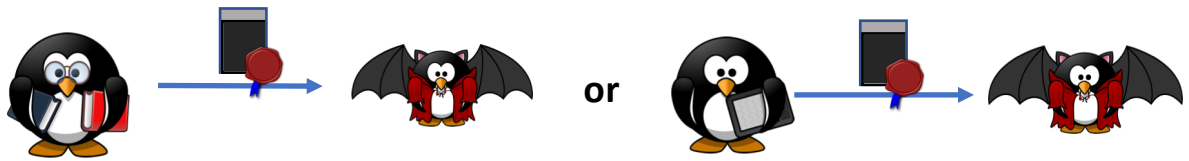
The user can hide irrelevant attributes. But the verifier can still check the validity of the credential.



Property: 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, try to distinguish users





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

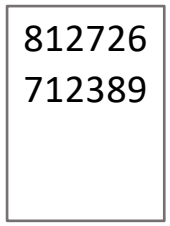
Why do traditional signatures not satisfy issuer unlinkability?

Construction 1: blind signatures

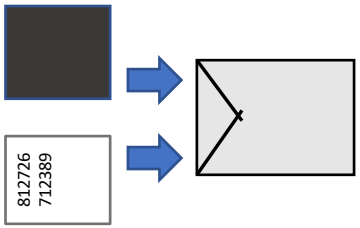
- Blind signature: signer signs a message *m* without knowing what it signs. Moreover, it cannot later recognize this signature.
- Property: exactly as issuer unlinkability.
- Implemented by: U-Prove by Microsoft.

A simple physical blind signature scheme

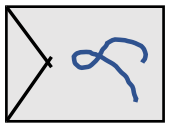
Step 1: write down serial number



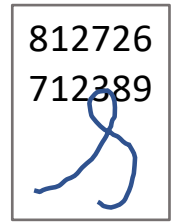
Step 2: place in envelope with carbon paper



Step 3: issuer signs the envelope

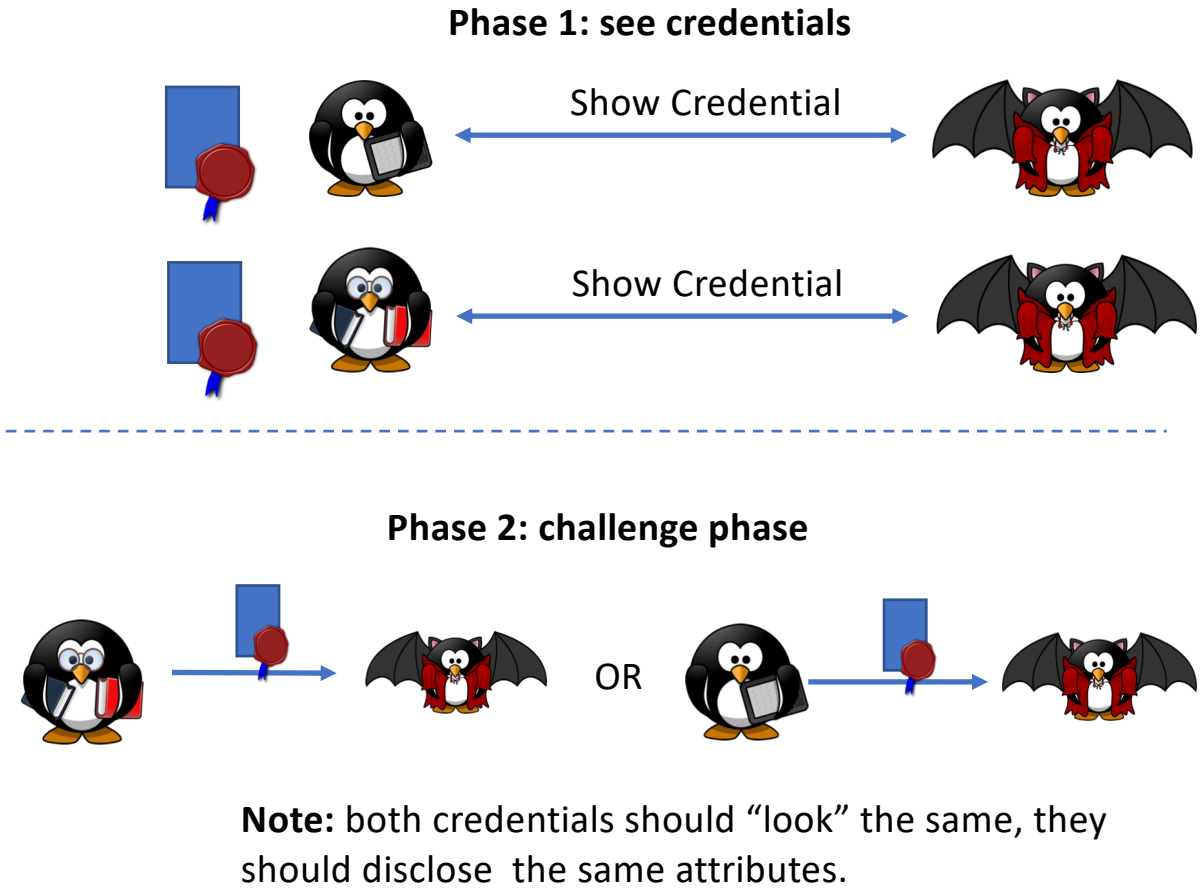


Step 4: user recovers signed statement



Property: 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, try to distinguish users



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

Construction 2: proving having signature

Issuing a credential

1. Commit to user-defined attributes

3. Return signature σ on attributes

Usually not a blind signature

2. Find other attributes

Showing a credential

Credential
Secret key
Age
Membership number

User

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

Verifier/SP

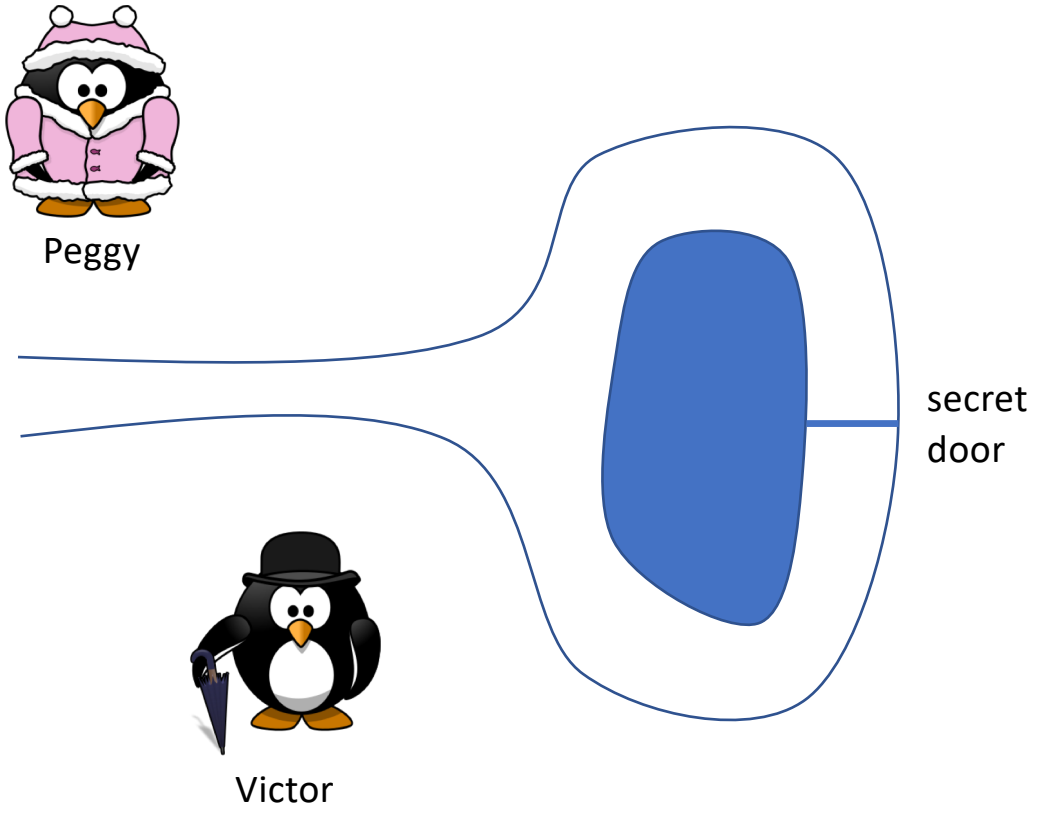
Intermezzo: zero-knowledge proofs

A famous example zero-knowledge proof

The cave of Ali Baba

Peggy (the prover) discovers a cave with a secret door. When she speaks the right password the door opens.

How can she convince Victor (the verifier) that she can open the door without revealing the password?

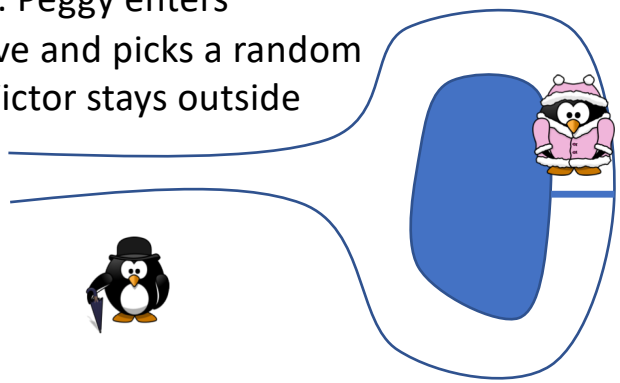


Proving you know the password

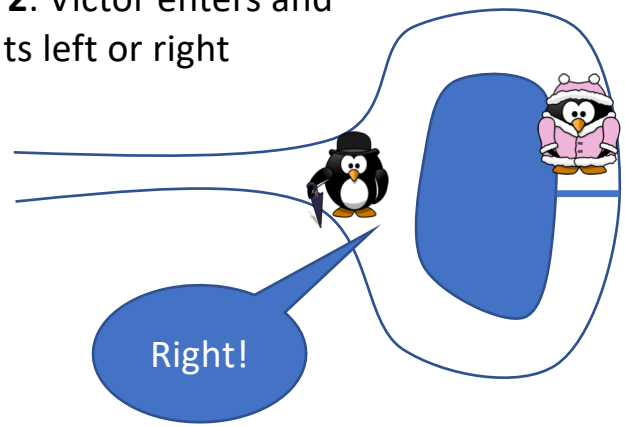
Peggy

Victor

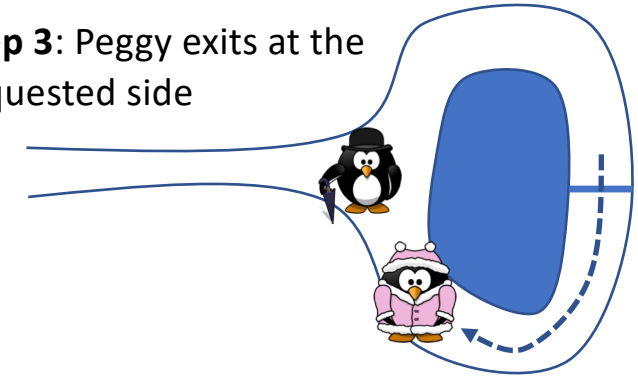
Step 1: Peggy enters the cave and picks a random side. Victor stays outside



Step 2: Victor enters and shouts left or right



Step 3: Peggy exits at the requested side



Step 4: Repeat!

Properties of zero-knowledge proofs

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 very high probability (i.e., very close to 1).

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

Proving zero-knowledge property

For every verifier there exists some **simulator** that given the validity of the statement (and without interacting with an honest prover) can produce **transcripts** that “look like” an interaction between the verifier and an honest prover.

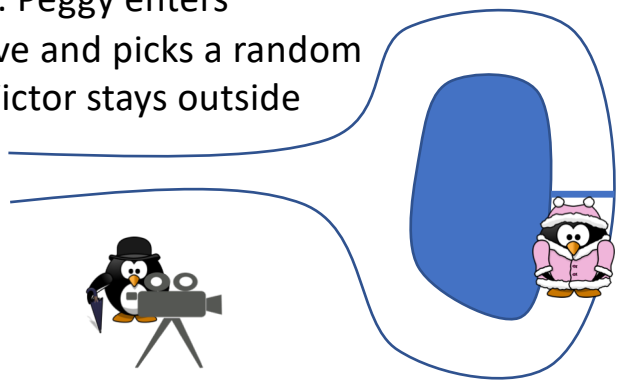
How do you show that there exists such a simulator. Let’s build one for Ali Baba’s cave.

Simulating Ali Baba's cave

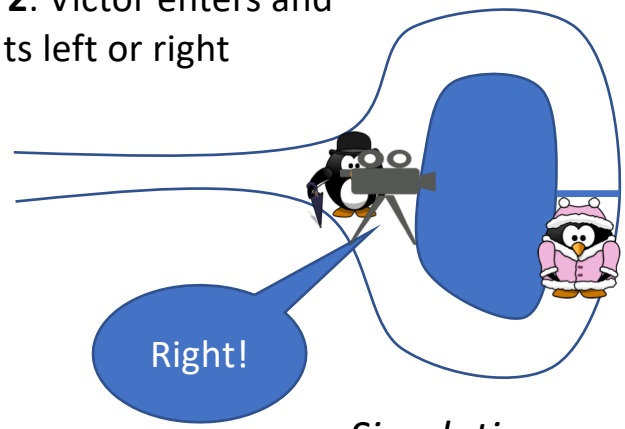
Peggy

Victor

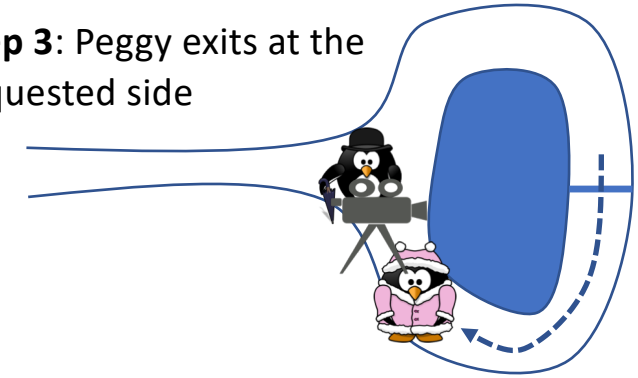
Step 1: Peggy enters the cave and picks a random side. Victor stays outside



Step 2: Victor enters and shouts left or right



Step 3: Peggy exits at the requested side

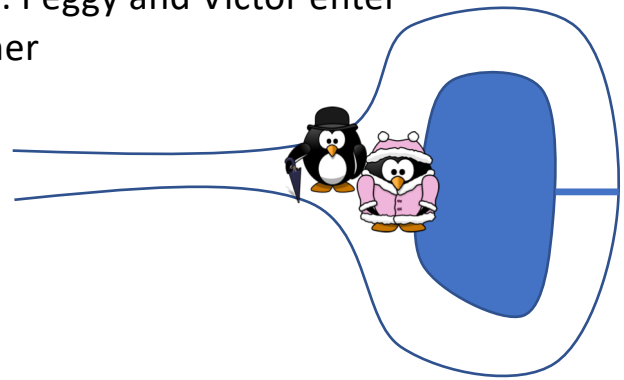


Simulation:

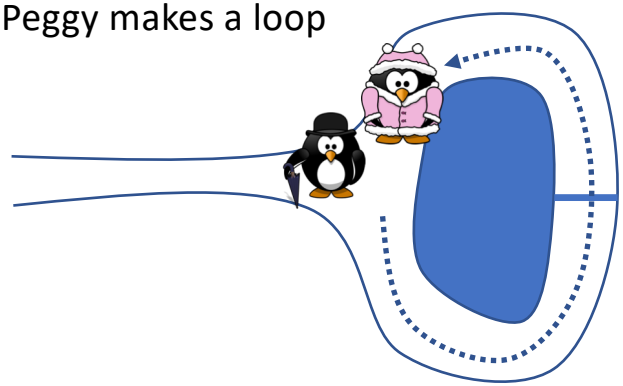
1. *Victor and Peggy agree on order: right, right, left, right, etc. before starting the recording*
2. *For every run, Peggy enters the right branch*
3. *Recording looks same as if Peggy really proved it*

A not simulateable variant

Step 1: Peggy and Victor enter together



Step 2: Peggy makes a loop



Completeness and soundness are still satisfied.

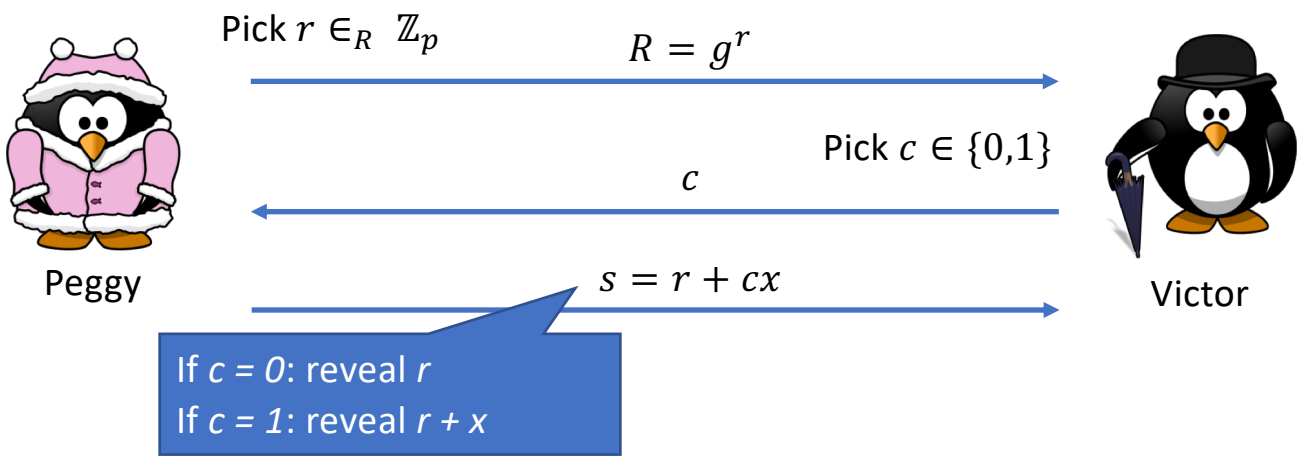
Zero-knowledge: cannot simulate this. Intuition: the recording allows Victor to convince other people. Hence, he learns more.

Schnorr's proof of identification (simplified)

Peggy wants to prove that she knows the private key x corresponding to the public key h.

Common information:

- Group: (G, g, p)
- Public key: h



Cryptography sidebar

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

Thus: $g^p = 1$

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

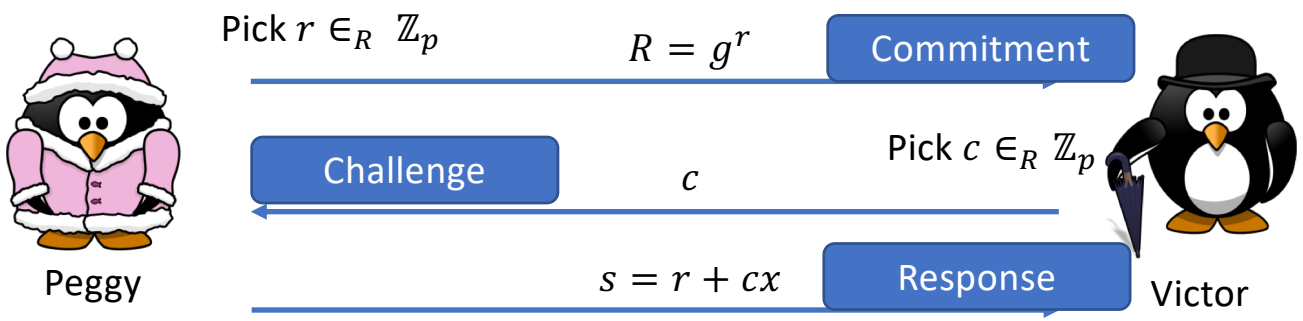
Verify:
If $c = 0$: $R = g^s$
If $c = 1$: $Rh = g^s$

Schnorr's proof of identification

Peggy wants to prove that she knows the private key x corresponding to the public key h.

Common information:

- Group: (G, g, p)
- Public key: h



Cryptography sidebar

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

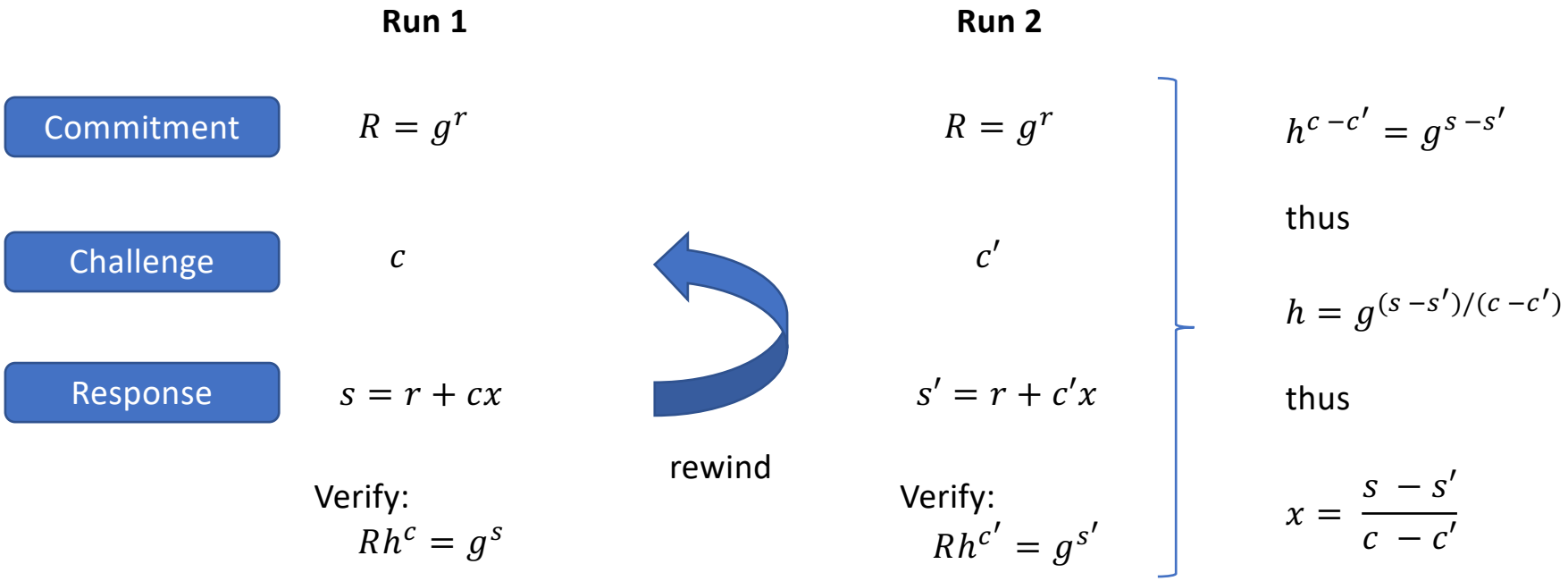
Thus: $g^p = 1$

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

Verify:
$Rh^c = g^s$

Proof of knowledge

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



Honest-verifier special soundness

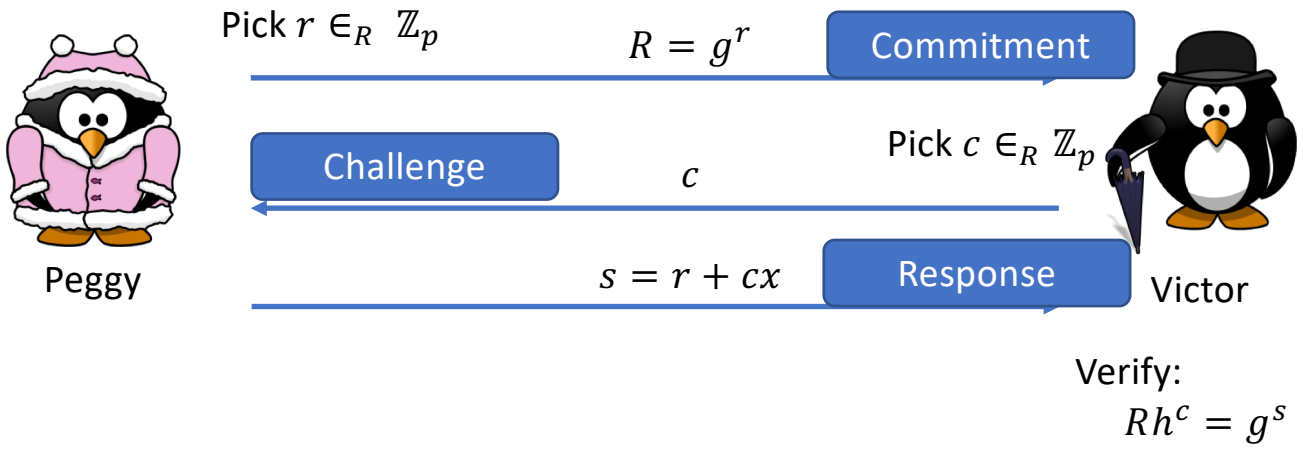
- Given two answers to a single commitment: can extract the secret.
- Hence, a cheating prover can answer at most one challenge.
- Soundness probability is:

$$1 - \frac{1}{p}$$

A shorthand notation

Common information:

- Group: (G, g, p)
- Public key: h



Cryptography sidebar

Cyclic group: G
Generator: g
Group order: p

Thus: $g^p = 1$

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

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

Other proofs of knowledge

- Proving equality of discrete logarithms:

$$\mathrm{PK}\{(x): h_1 = g_1^x \ \wedge h_2 = g_2^x\}$$

- Proving knowledge of a Pedersen's commitment:

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

Practical use of zero-knowledge proofs

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

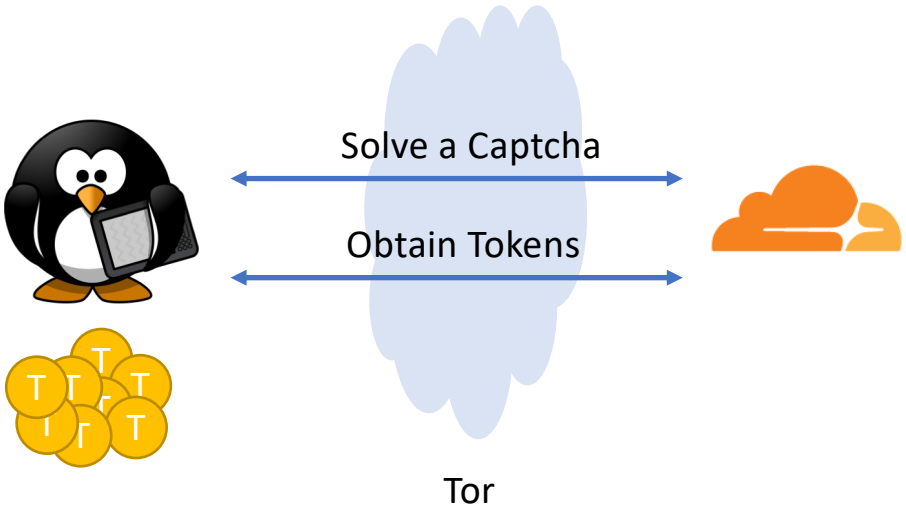
- Schnorr's signatures
- Zcash digital currency (but uses a different type of proofs)
- Other types of electronic cash based on tokens
- Electronic voting systems
- Private (smart) metering
- Privacy-friendly reputation systems

Example: privacy pass

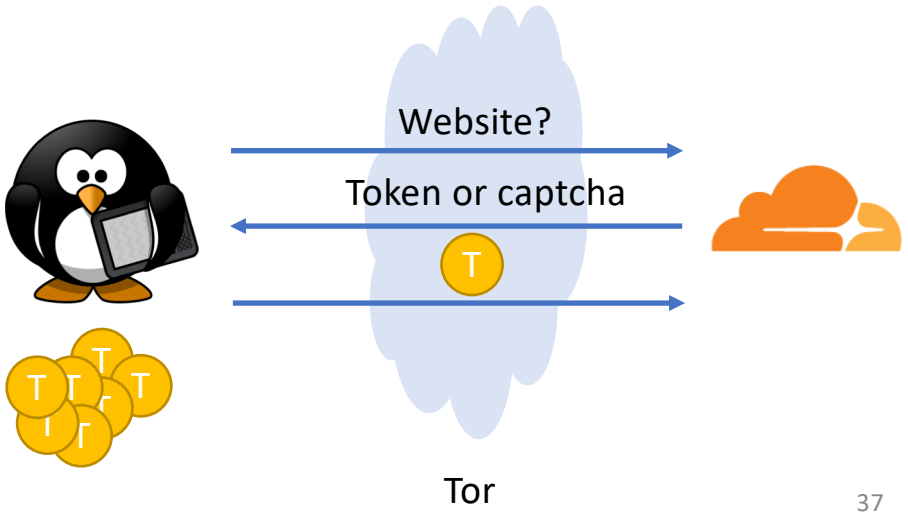
- Goal: automatically allow real users, implemented by Cloudflare and Tor browser.

How would you implement the tokens?

Step 1: Obtain tokens

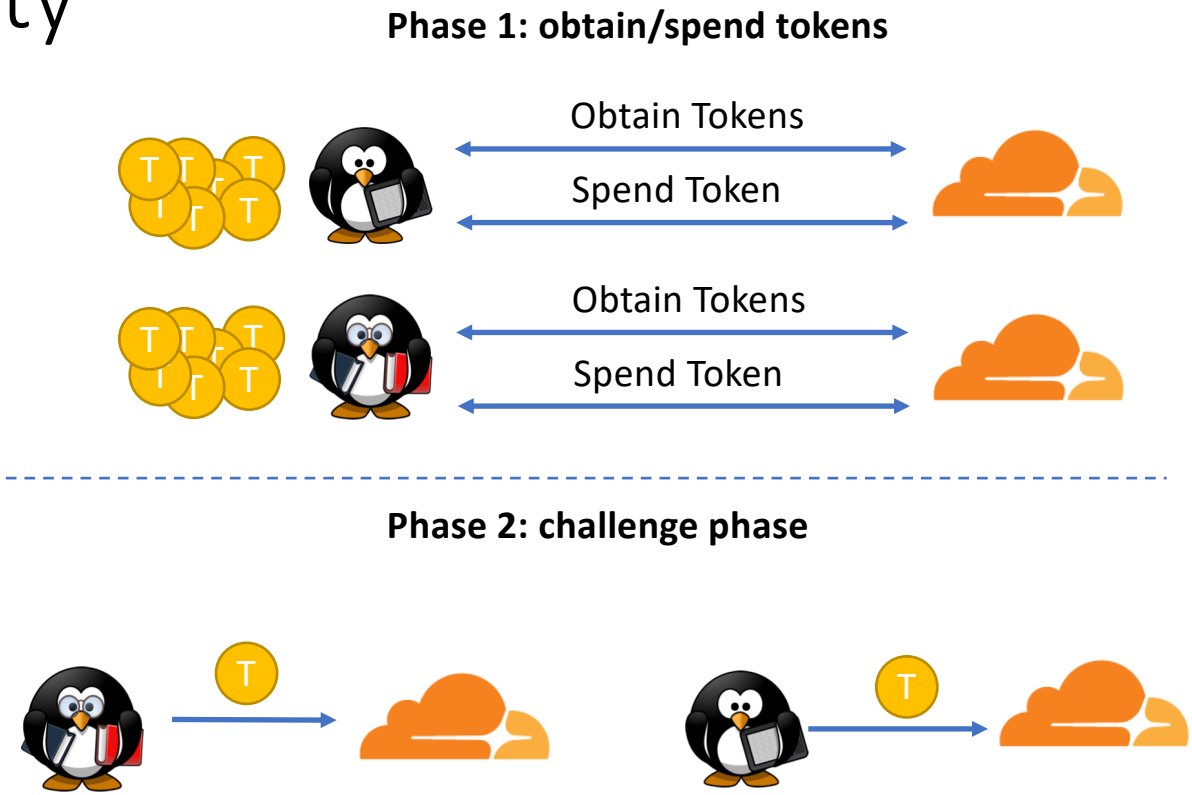


Step 2: Spend tokens instead of solving captcha



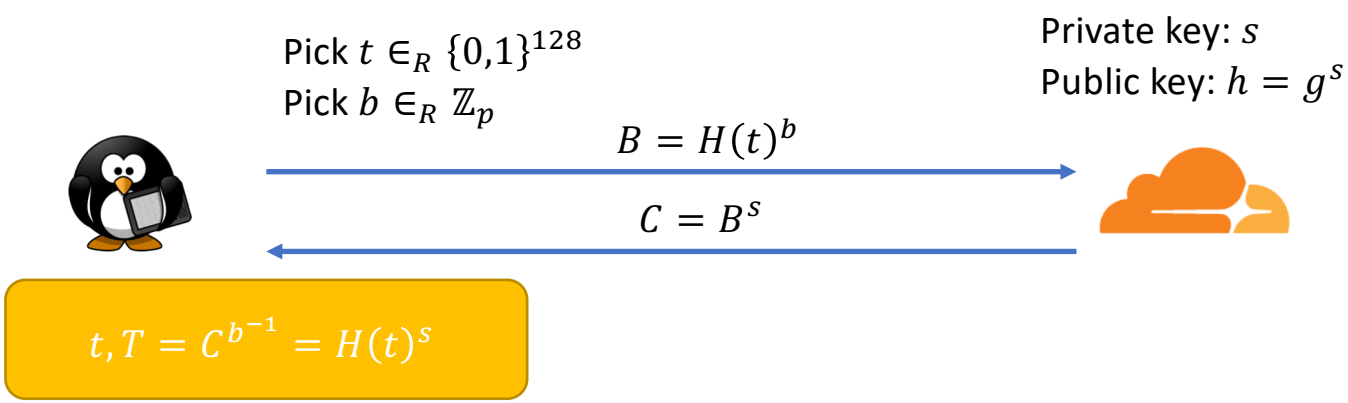
Property: unlinkability

- Cloudflare (the adversary) should not be able to link tokens by the same user.
- Modelled using an indistinguishability game.
- Phase 1: obtain/spend tokens
- Phase 2: challenge phase, distinguish users

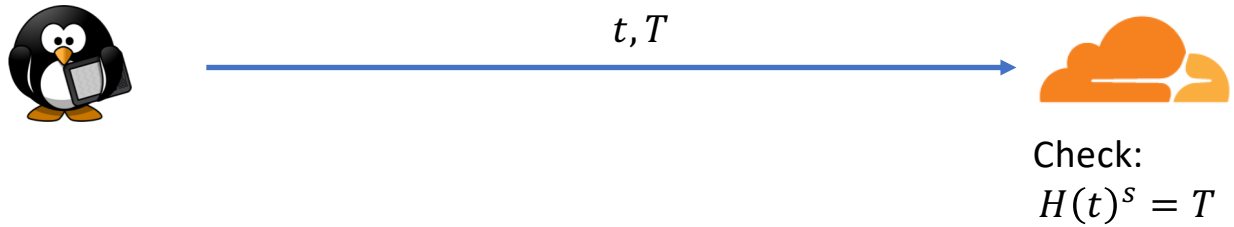


Implementing privacy pass

Obtaining a token



Showing a token



Cryptography sidebar

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

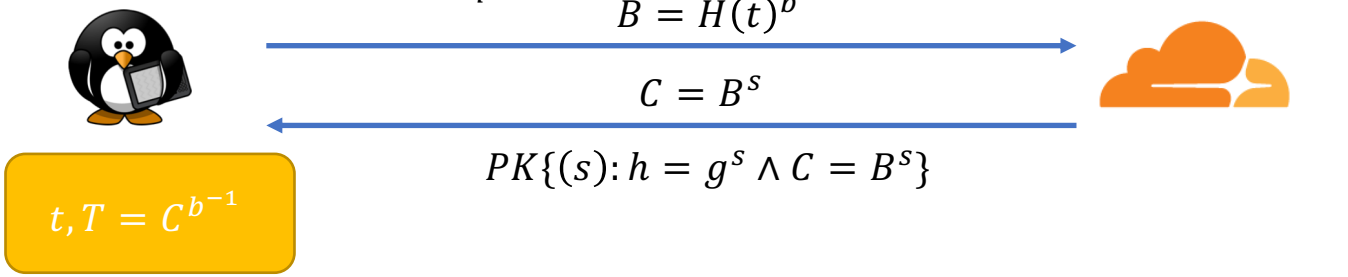
Thus: $g^p = 1$

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

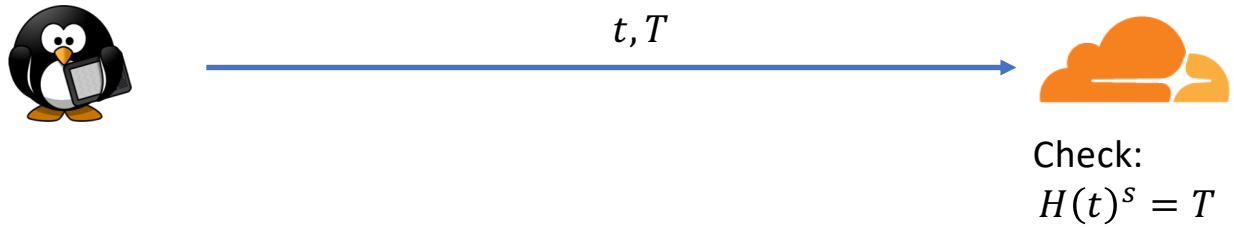
Hash function: $H: \{0,1\}^* \rightarrow G$

Implementing privacy pass

Obtaining a token



Showing a token



Cryptography sidebar

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

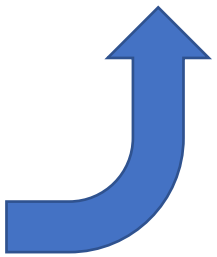
Thus: $g^p = 1$

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

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

Honest but curious versus malicious

- Honest but curious model (passive)
  - Common in cryptographic protocol
  - Assumes that parties follow the protocol as prescribed
  - But they might try to learn as much as they can from what they can see
  - Example: Yao's garbled circuit (*What do we trust the parties for?)*
- Malicious model (active)
  - Parties may arbitrarily deviate from protocol
  - May learn much more by deviating

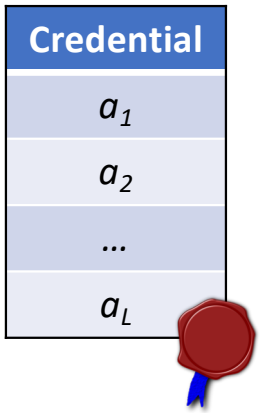


Common trick:
Add zero-knowledge proofs to ensure malicious parties behave honestly.

An example credential scheme

Basic setting

- L attributes
- Some of these determined by issuer
- Some of these determined by the user



BBS+ credentials: a signature scheme

Cryptography sidebar

Cyclic groups: G_1, G_2, G_T
Generators: g, h, g_T
Group order: p

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

Key generation

- Private key $\gamma \in_R \mathbb{Z}_p$, public key $w = h^\gamma$
- Generators: $B_0, B_1, \ldots, B_L \in_R G_1$

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

- Pick $e, s \in_R \mathbb{Z}_p$, and set
- $A = \left(g\, B_0^s \prod_{i=1}^{L} B_i^{m_i}\right)^{\frac{1}{e+\gamma}}$
- Signature: (A, e, s)

Verifying a signature:

- Given a signature (A, e, s)
- And public key $w, B_0, \ldots, B_L$
- Check:

$e(A, wh^e) = e(gB_0 \prod_{i=1}^{L} B_i^{m_i}, \mathrm{h})$

BBS+: the name comes from “Boneh-Boyen signature”

Issuing a credential

- User commits to hidden attributes:

$$C = B_0^{s'} \prod_{i \in H} B_i^{a_i}$$

- And proves that she did so correctly:

$$PK\left\{(s', (a_i)_{i \in H}) : C = B_0^{s'} \prod_{i \in H} B_i^{a_i}\right\}$$

- Issuer verifies the proof, picks e, s'' and sets:

$$A = \left(g\, C B_0^{s''} \prod_{i \in D} B_i^{a_i}\right)^{\frac{1}{e+\gamma}}$$

- User forms signature $(A, e, s) = (A, e, s' + s'')$

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

Cryptography sidebar

Cyclic groups: G_1, G_2, G_T
Generators: g, h, g_T
Group order: p

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

BBS+ Signatures
Secret key: $\gamma \in_R \mathbb{Z}_p$
Public key: $w = h^\gamma$
Generators: $B_0, B_1, \ldots, B_L \in_R G_1$

Signature: (A, e, s) such that

$$A = \left(g\, B_0^{s} \prod_{i=1}^{L} B_i^{a_i}\right)^{\frac{1}{e+\gamma}}$$

A simplified notation: BBS+ credential

- Given a credential (A, e, s)
- Hidden attributes: $(a_i)_{i \in H}$ (not shown to verifier)
- Disclosed attributes: $(a_i)_{i \in D}$ (shown to verifier)
- Construct the "proof":

$$PK\left\{(A, e, s, (a_i)_{i \in H}) : A^{e+\gamma} = gB_0^s \prod_{i \in D} B_i^{a_i} \prod_{i \in H} B_i^{a_i}\right\}$$

(instantiation is more complex)

Cryptography sidebar

Cyclic groups: G_1, G_2, G_T
Generators: g, h, g_T
Group order: p

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

BBS+ Signatures
Secret key: $\gamma \in_R \mathbb{Z}_p$
Public key: $w = h^\gamma$
Generators: $B_0, B_1, \ldots, B_L \in_R G_1$

Signature: (A, e, s) such that

$$A = \left(g\, B_0^s \prod_{i=1}^{L} B_i^{a_i}\right)^{\frac{1}{e+\gamma}}$$

The real deal

Scary stuff. Shown for completeness

- Given a credential (A, e, s)
- Create commitments:

$$C_1 = A g_2^{r_1} \quad \text{and} \quad C_2 = g_1^{r_1} g_2^{r_2}$$

- Construct proof of knowledge:

$$\mathrm{PK}\Big\{ (r_1, r_2, \alpha_1, \alpha_2, e, (a_i)_{i\in\mathcal{H}}, s) : C_2 = g_1^{r_1} g_2^{r_2} \wedge C_2^e = g_1^{\alpha_1} g_2^{\alpha_2} \ \wedge$$

$$\hat{e}(C_1, w)\hat{e}(C_1, h)^e = \hat{e}(g, h)\hat{e}(B_0, h)^s$$

$$\prod_{i\in\mathcal{H}} \hat{e}(B_i, h)^{a_i} \prod_{i\in\mathcal{D}} \hat{e}(B_i, h)^{a_i} \hat{e}(g_2, w)^{r_1} \hat{e}(g_2, h)^{\alpha_1} \Big\},$$

Cryptography sidebar

Cyclic groups: G_1, G_2, G_T
Generators: g, h, g_T
Group order: p

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

BBS+ Signatures
Secret key: $\gamma \in_R \mathbb{Z}_p$
Public key: $w = h^\gamma$
Generators: $B_0, B_1, \ldots, B_L \in_R G_1$

Signature: (A, e, s) such that

$$A = \left(g\, B_0^s \prod_{i=1}^{L} B_i^{m_i} \right)^{\frac{1}{e+\gamma}}$$

Properties of BBS+ credentials

- **Unforgeability**: yes, from the BBS+ signatures. They rely on the *q*-SDH assumption.
- **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 commitments and the proof of knowledge hide the signature. Therefore, neither the signer nor the verifier can recognize it.

Other credential schemes

- U-Prove by Stefan Brands/Microsoft around 2000
  - Based on the blind signing paradigm
  - Standard discrete-logarithm based setting, no pairings
  - To get unlinkability: use a credential only once
- Identity Mixer (Idemix) by Jan Camenisch and Anna Lysyanskaya/IBM research around 2004
  - 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

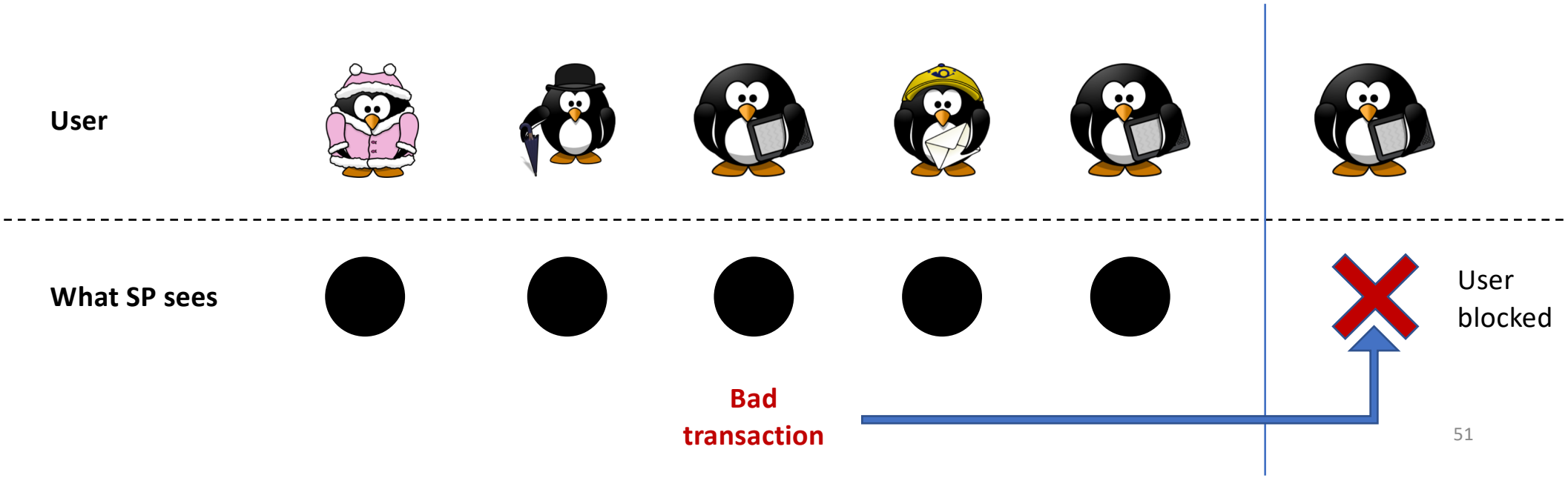
What if things go wrong?

Revoking a credential

- To revoke 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? What information is needed?
  - Can users detect that they have been revoked? (Or can the revocation test be made in silence?)
  - Does the system provide backward unlinkability after revocation?

Blacklistable anonymous credentials

- What if we do *not* know the user's identity?
- For the revocation schemes so far we need to know or recover the user's identity in order to revoke a credential.

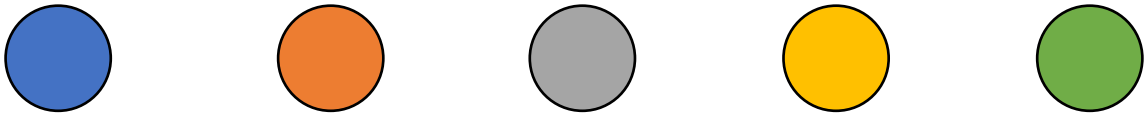


Blacklistable anonymous credentials, idea

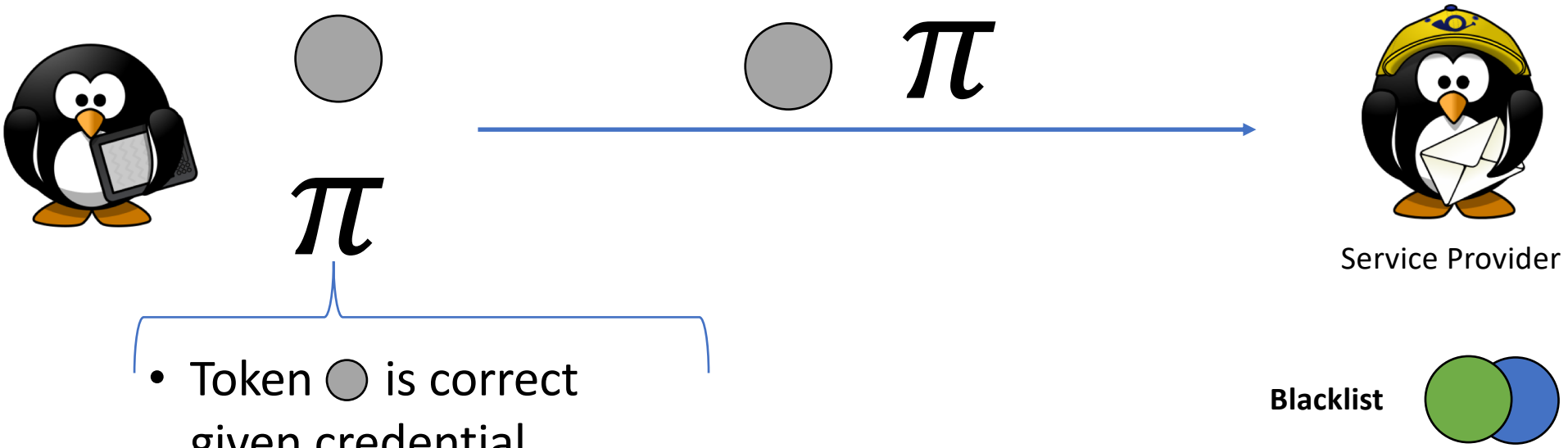
- For every transaction, user produces a token
- Tokens belong to users, but SP cannot determine which user
- Users use a credential to prove that the token is correctly formed

User

Token for SP



Blacklistable anonymous credentials, idea II



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

Constructing tokens and proofs

- Every user has a credential (A, e, s), 0 attributes
- Tokens: $(h, t = h^s)$ where h a random element
- Blacklist: $BL = \{(h_1, t_1), \ldots, (h_n, t_n)\}$
- Construct the proof:

$$PK\left\{(A, e, s): \begin{array}{l} A^{e+\gamma} = gB_0^s \ \wedge \\ t = h^s \ \wedge \\ \bigwedge_{i=1}^{n} (t_i \neq h_i^s) \end{array}\right.$$

Have credential

Token correct

Blacklisted tokens not mine

Cryptography sidebar

Cyclic groups: G_1, G_2, G_T
Generators: g, h, g_T
Group order: p

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

BBS+ Signatures
Secret key: $\gamma \in_R \mathbb{Z}_p$
Public key: $w = h^\gamma$
Generators: $B_0, B_1, \ldots, B_L \in_R G_1$

Signature: (A, e, s) such that

$$A = \left(g\, B_0^s \prod_{i=1}^{L} B_i^{a_i}\right)^{\frac{1}{e+\gamma}}$$

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
- 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:

- Issuer can trace users without their knowledge
- Does not give backward unlinkability

Alternative: use accumulators to hold a blacklist. Does not suffer from backward unlinkability.

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

Summary of mitigation mechanisms

- Blacklistable anonymous credentials (BLAC): block misbehaving anonymous users without needing to identify them
- *n*-times anonymous credentials: limit how often a credential can be shown without it becoming linkeable.
- Simple revocation mechanisms based on tokens a a user-held or issuer-held secret. Suffers from backward linkability problems.

References

- Nigel Smart. Cryptography: An Introduction. 3rd edition online. Also available in print. Chapters 11 and 25.
- https://privacypass.github.io/protocol/
- Man Ho Au, Willy Susilo, and Yi Mu. “Constant-Size Dynamic k-TAA”. In: SCN 2006. (BBS+ credentials)
- 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.

Using accumulators for revocation

- An accumulator is a compact data structure representing a list. For every item in the accumulator, there is a witness that proves that the item is in the accumulator.
- Can construct zero-knowledge proofs of having a witness
- Use accumulators to construct:
  - Blacklist: all revoked credentials
  - Whitelist: all unrevoked credentials

Properties

- Backward unlinkability
- Constant size proofs (but more complicated than blacklistable credentials)

Construction 1: 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.

U-Prove issuance:

- User has secret key *sk*
- Issuer has attributes $(a_1,\ldots a_k)$
- Issuer blindly signs m = $(sk, a_1,\ldots a_k)$
- User holds credential (σ, m)

U-Prove showing:

- User sends signature σ and message m*