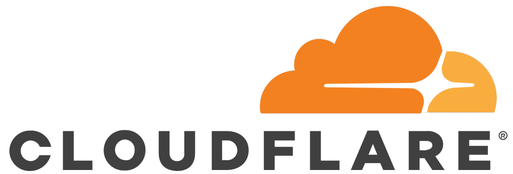
Post-Quantum Privacy Pass

Via Post-Quantum Anonymous Credentials

Guru Vamsi Policharla, Bas Westerbaan, Armando Faz Hernández, and Chris Wood



RWC 2023

Post-Quantum Crypto Today

Post-Quantum Crypto Today

Encryption	Signatures
😀	😀
CRYSTALS-KYBER	CRYSTALS-Dilithium FALCON SPHINCS+

Post-Quantum Crypto Today

Encryption	Signatures	*Advanced* Crypto
		
CRYSTALS-KYBER	CRYSTALS-Dilithium FALCON SPHINCS+	Blind Sigs - *semi-practical* OPRFs - *semi-practical* Anon. Creds - theoretical

The need for *efficient* PQ Blind Sigs/AC/OPRF

… and many more!

The need for *efficient* PQ Blind Sigs/AC/OPRF

Impacts billions of users!

… and many more!

Post-Quantum Crypto Today

Encryption	Signatures	*Advanced* Crypto
😃	😃	☹️
CRYSTALS-KYBER	CRYSTALS-Dilithium FALCON SPHINCS+	Blind Sigs - *semi-practical* OPRFs - *semi-practical* Anon. Creds - ~~theoretical~~ *semi-practical*

Blind Signatures [Cha82]

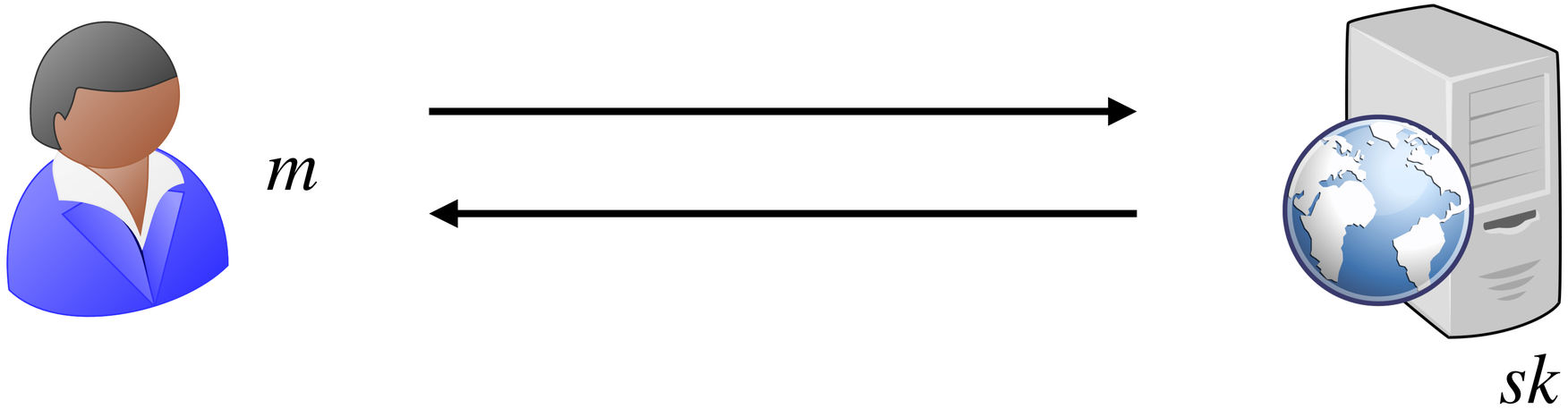
m

sk

Goal — Alice obtains a signature such that:

- Server learns nothing about m
- Alice learns nothing about sk

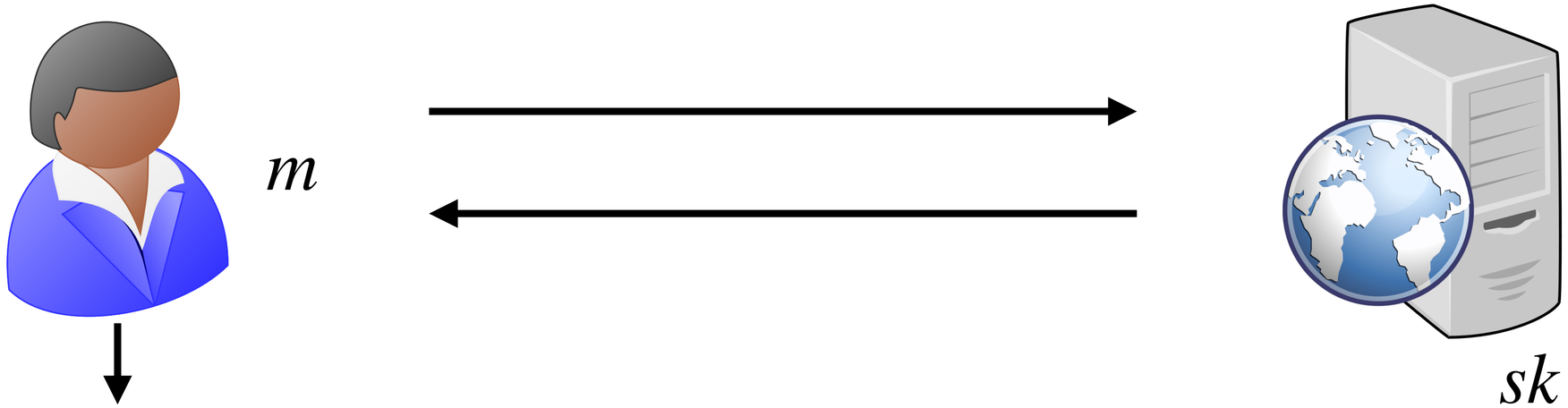
Blind Signatures [Cha82]



Goal — Alice obtains a signature such that:

- Server learns nothing about m
- Alice learns nothing about sk

Blind Signatures [Cha82]



$\sigma \mid \text{Verify}_{vk}(m, \sigma) = 1$

Goal — Alice obtains a signature such that:

- Server learns nothing about m
- Alice learns nothing about sk

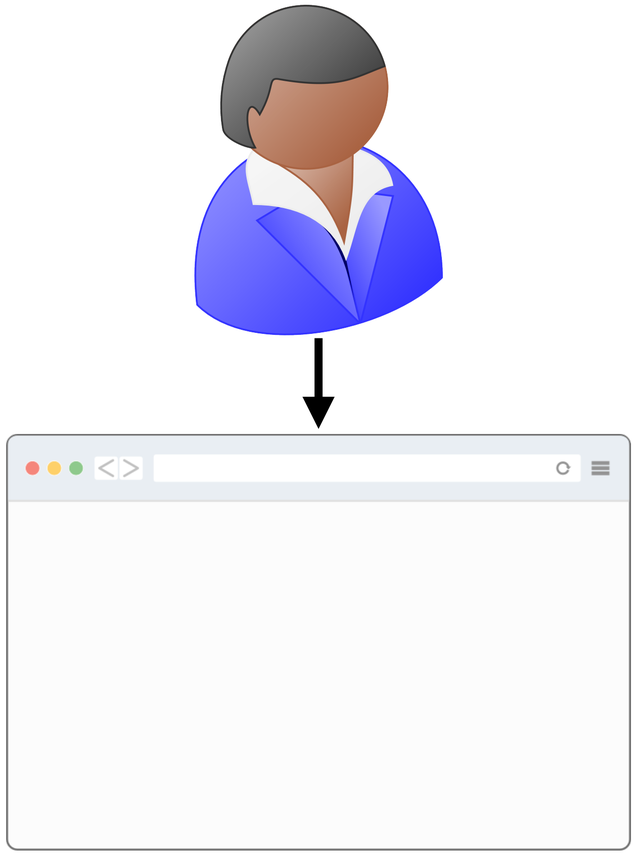
How are Blind Signatures used?

How are Blind Signatures used?

Commonly used to authenticate in a privacy preserving manner

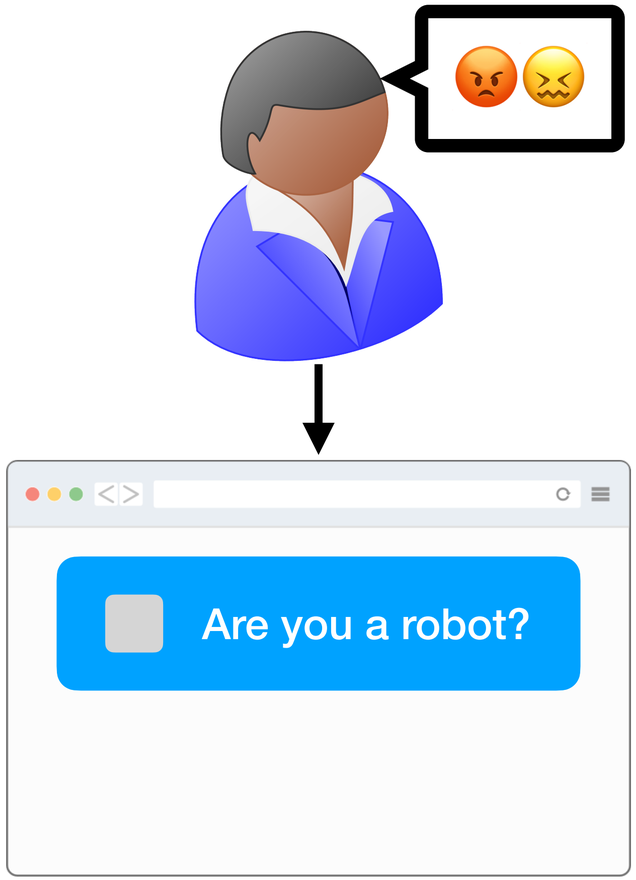
How are Blind Signatures used?

Commonly used to authenticate in a privacy preserving manner



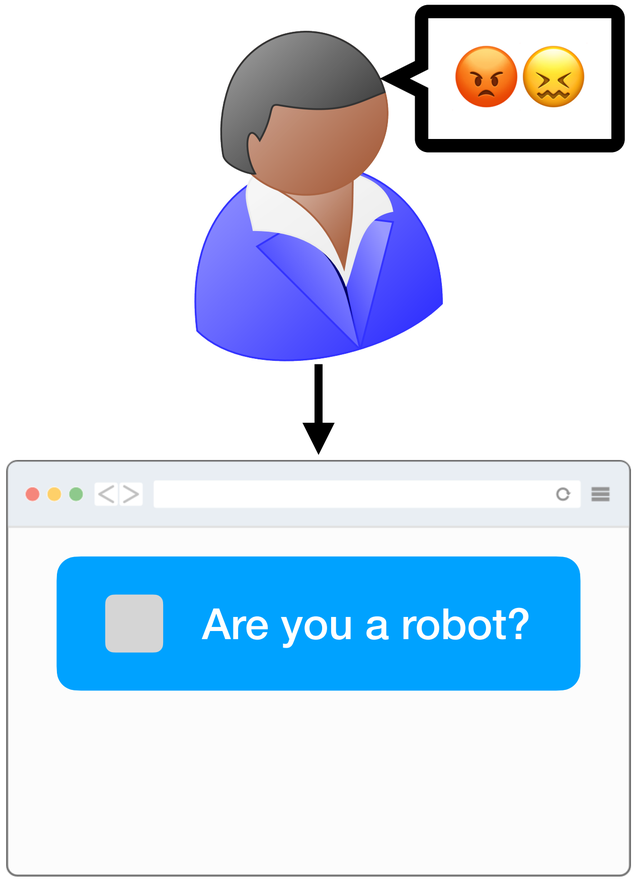
How are Blind Signatures used?

Commonly used to authenticate in a privacy preserving manner



How are Blind Signatures used?

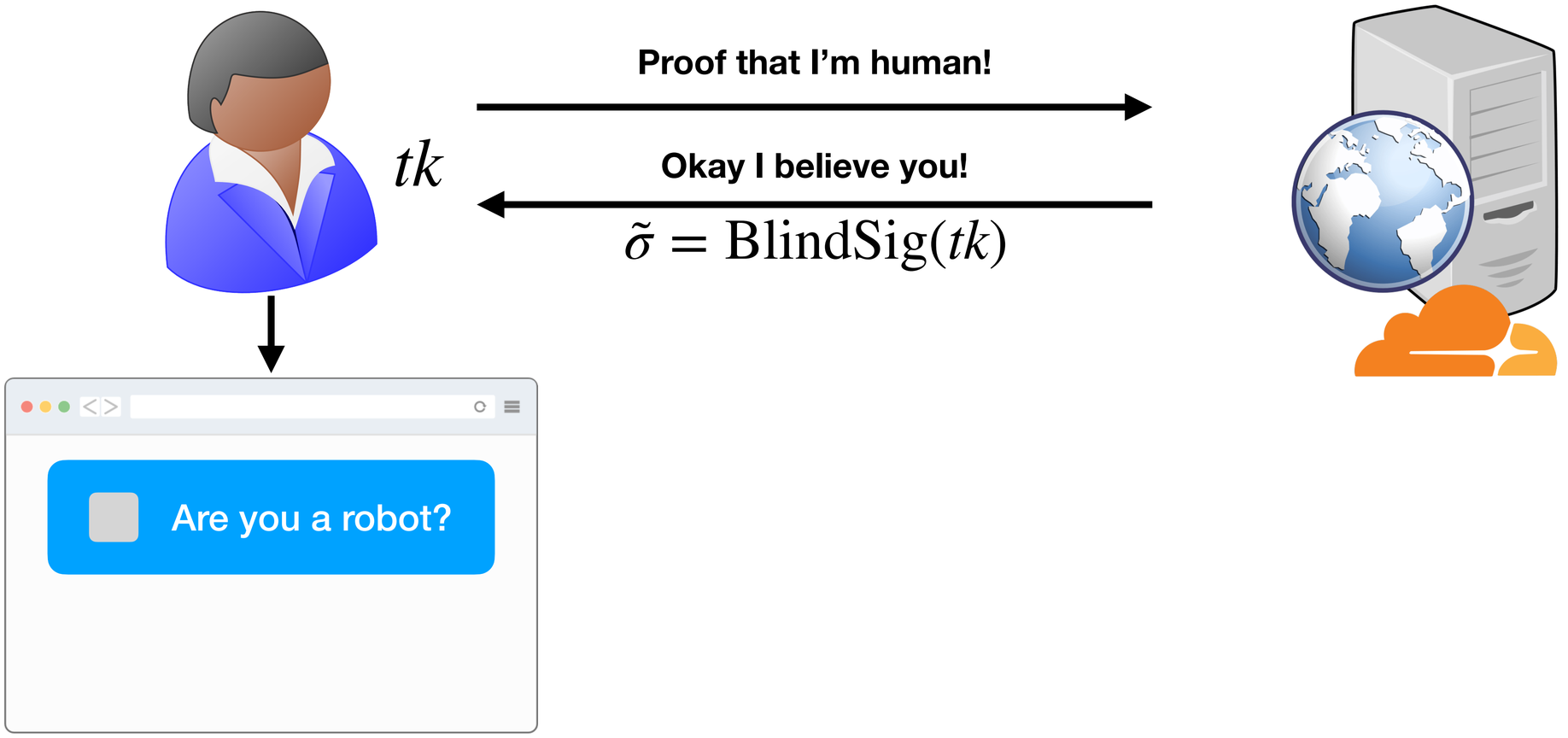
Commonly used to authenticate in a privacy preserving manner



Can we avoid CAPTCHAs without compromising privacy?

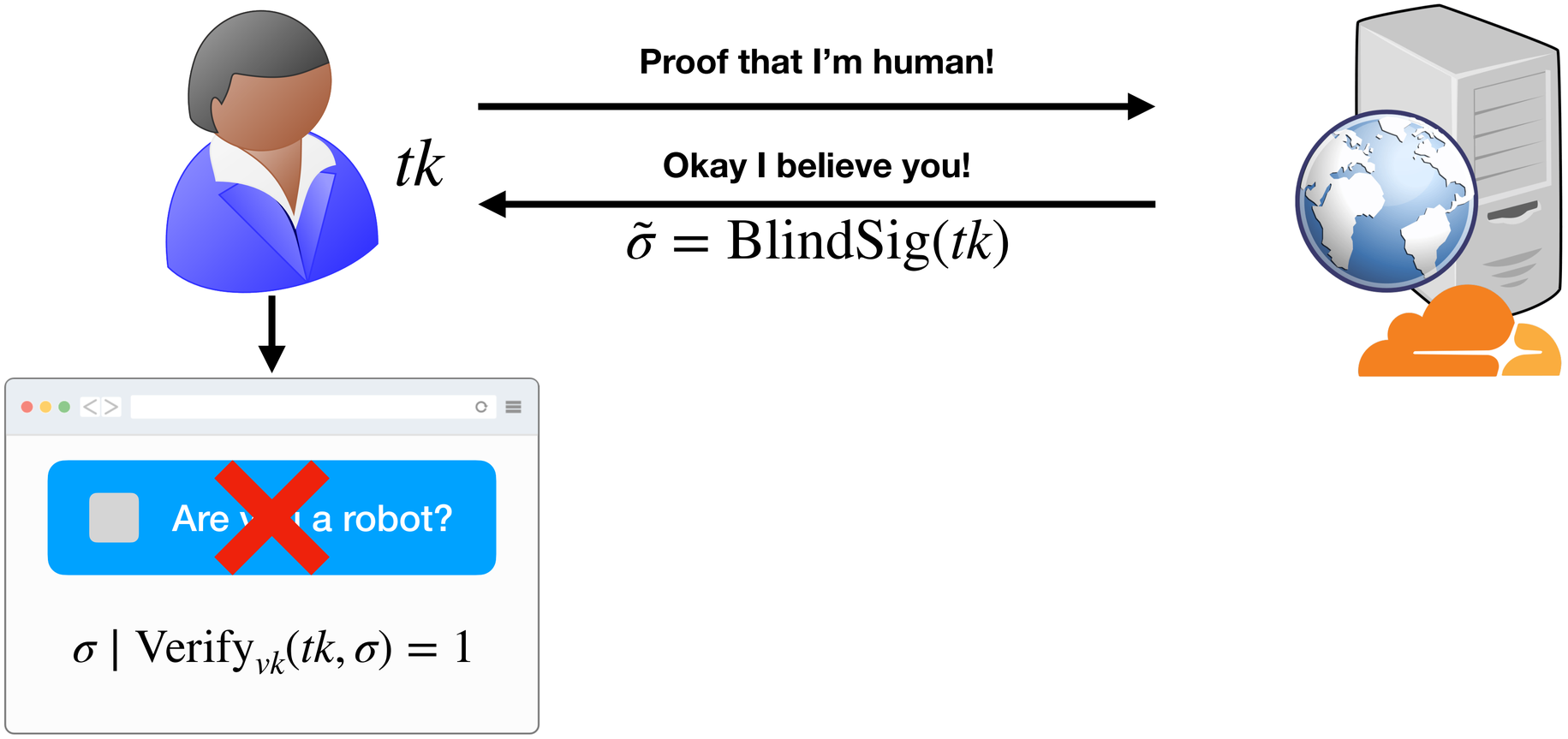
How are Blind Signatures used?

Commonly used to authenticate in a privacy preserving manner



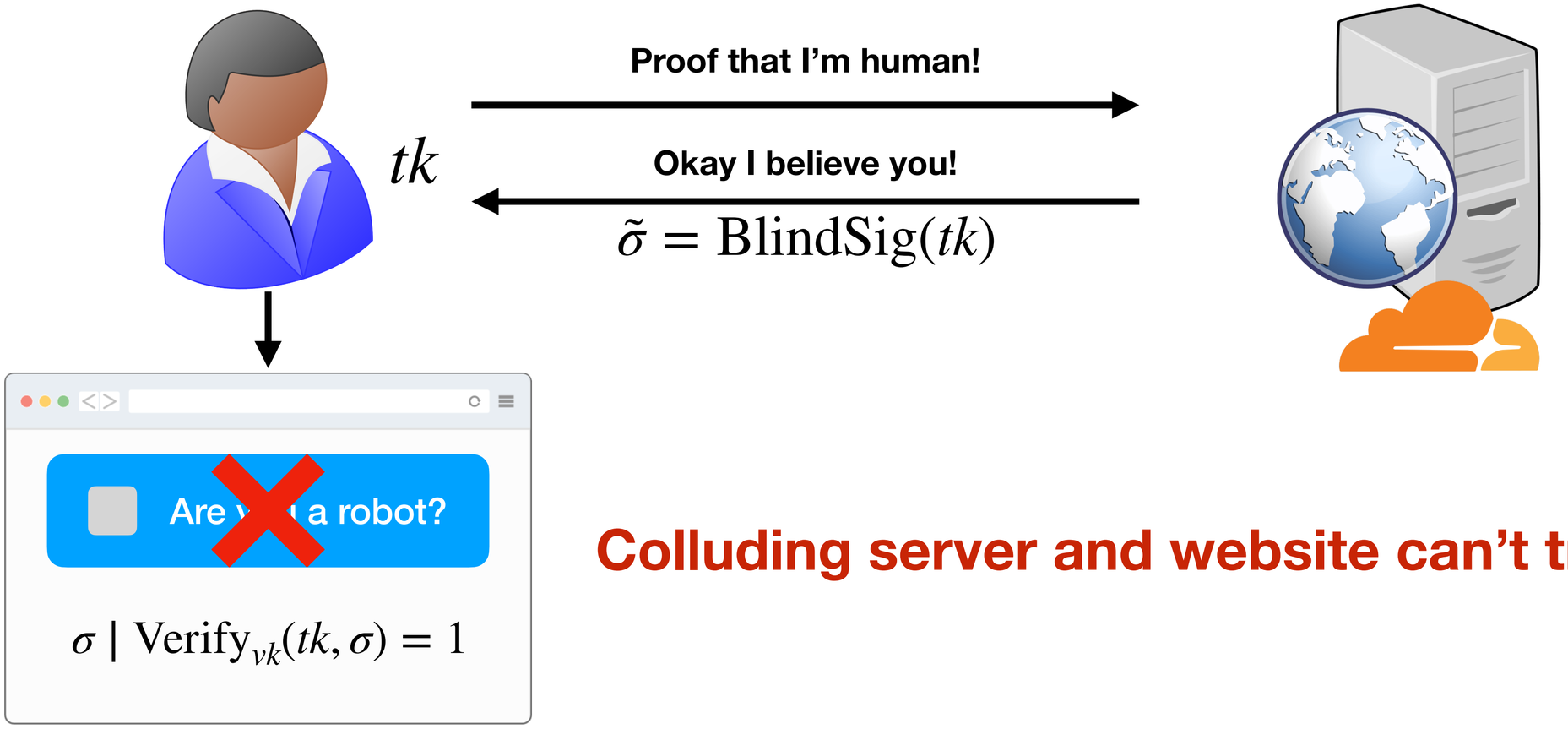
How are Blind Signatures used?

Commonly used to authenticate in a privacy preserving manner



How are Blind Signatures used?

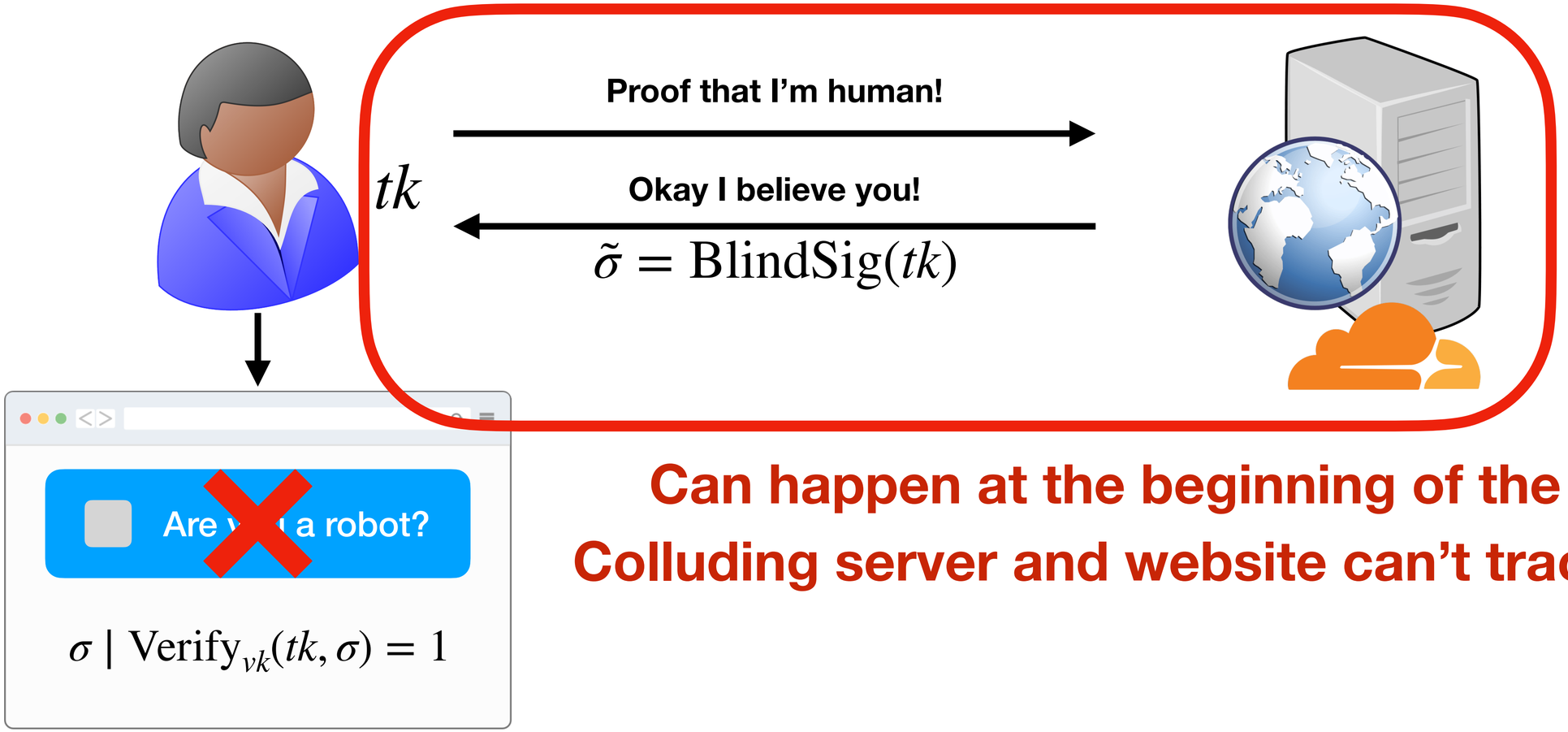
Commonly used to authenticate in a privacy preserving manner



Colluding server and website can't track a user!

How are Blind Signatures used?

Commonly used to authenticate in a privacy preserving manner

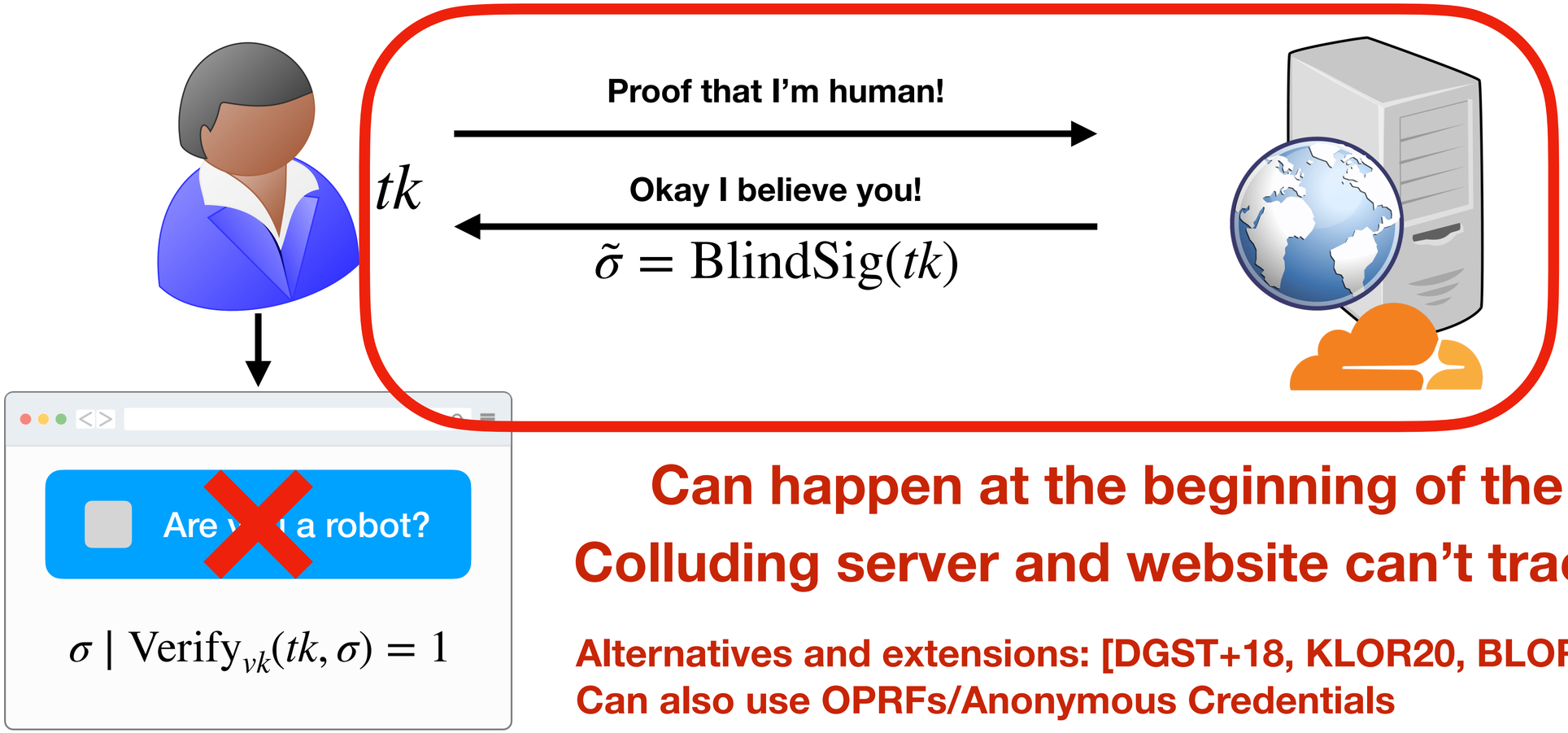


Can happen at the beginning of the day!

Colluding server and website can't track a user!

How are Blind Signatures used?

Commonly used to authenticate in a privacy preserving manner



Can happen at the beginning of the day!

Colluding server and website can't track a user!

Alternatives and extensions: [DGST+18, KLOR20, BLOR22]
Can also use OPRFs/Anonymous Credentials

Building a PQ Blind Signature

[Fis06]

m

$c = \mathrm{com}(m; r)$

sk

Building a PQ Blind Signature

[Fis06]

m

$c = \mathrm{com}(m; r)$

$\sigma = \mathrm{Sign}(c)$

sk

Building a PQ Blind Signature

[Fis06]

m

$c = \mathrm{com}(m; r)$

$\sigma = \mathrm{Sign}(c)$

sk

Final signature is a zk proof of knowledge*:

- I know a signature on some commitment c
- I know an opening r of the commitment c to m

$$\Pi = \{\sigma, r, c \mid \mathrm{Verify}_{vk}(\sigma, c) \wedge m = \mathrm{open}(c; r)\}$$

** Extractor needs to be able to extract from many instances for security reduction*

Building a PQ Blind Signature

[Fis06]

m

$c = \mathrm{com}(m; r)$

$\sigma = \mathrm{Sign}(c)$

sk

Final signature is a zk proof of knowledge*:

- I know a signature on some commitment c
- I know an opening r of the commitment c to m

$$\Pi = \{\sigma, r, c \mid \mathrm{Verify}_{vk}(\sigma, c) \wedge m = \mathrm{open}(c; r)\}$$

Can also get Anonymous Credentials with $m = (att_1, \ldots, att_n)$!

** Extractor needs to be able to extract from many instances for security reduction*

Building a PQ Blind Signature

[Fis06]

m

$c = \mathrm{com}(m; r)$

$\sigma = \mathrm{Sign}(c)$

sk

Final signature is a zk proof of knowledge*:

- I know a signature on some commitment c
- I know an opening r of the commitment c to m

How do we instantiate this?

$$\Pi = \{\sigma, r, c \mid \mathrm{Verify}_{vk}(\sigma, c) \wedge m = \mathrm{open}(c; r)\}$$

Can also get Anonymous Credentials with $m = (att_1, \ldots, att_n)$!

** Extractor needs to be able to extract from many instances for security reduction*

Prior Work

General purpose proofs are thought to be too big and slow

“We expect the prover runtime to be at least 1 hour”— [AKSY22]

Prior Work

General purpose proofs are thought to be too big and slow

"We expect the prover runtime to be at least 1 hour"— [AKSY22]

Strategy: Move expensive parts "outside" statement. Quite non-trivial!
[AKSY22, dPK22, BLNS22]

Prior Work

General purpose proofs are thought to be too big and slow

"We expect the prover runtime to be at least 1 hour"— [AKSY22]

Strategy: Move expensive parts "outside" statement. Quite non-trivial!
[AKSY22, dPK22, BLNS22]

Caveat: Cannot be used to directly get Anonymous Credentials

Prior Work

General purpose proofs are thought to be too big and slow

"We expect the prover runtime to be at least 1 hour"— [AKSY22]

Strategy: Move expensive parts "outside" statement. Quite non-trivial!
[AKSY22, dPK22, BLNS22]

Caveat: Cannot be used to directly get Anonymous Credentials

What if we carefully choose signature+proof system and optimize?
How expensive are the Anonymous Credentials?

Our PQ Blind Signature / AC

	Client work (s)	Verification (ms)	Size (KB)
Better time	~0.3	32	174
Balanced	~0.6	22	113
Better size	~4.8	20	86

Surprisingly efficient! Sizes comparable to Blind Signatures.

Strategy Overview

Two pronged effort

Strategy Overview

Two pronged effort

- Dilithium → zkDilithium:
  - Modify to make ZKP “friendly”
  - Use ZKP friendly hashes (Poseidon [GKRRM21])
- zkSTARK proof system [BBHR18]:
  - Match the field with zkDilithium
  - Reduce zkDilithium verification to simpler circuits

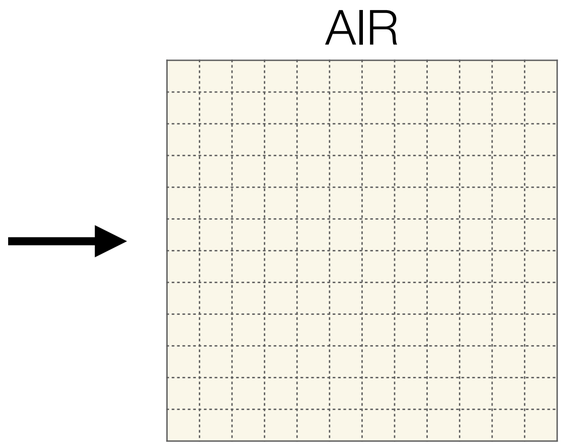
Some details

Pipeline of zkSTARKs

```
fn main() {
    // imagine complicated logic
    // ...
    // ...
    // ...

    println!("Hello, world!");
}
```

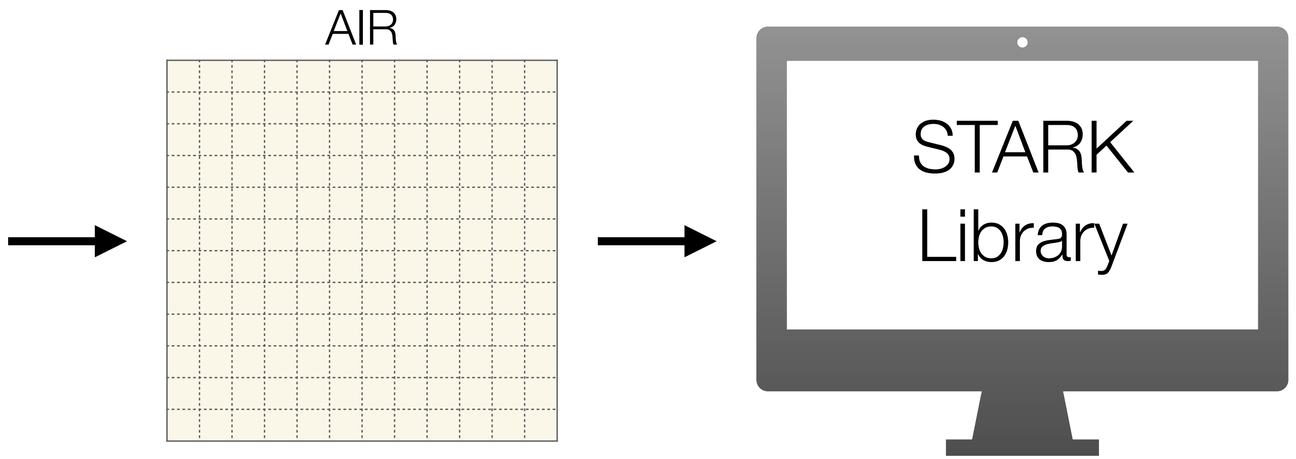
Pipeline of zkSTARKs

```rust
fn main() {
    // imagine complicated logic
    // ...
    // ...
    // ...

    println!("Hello, world!");
}
```



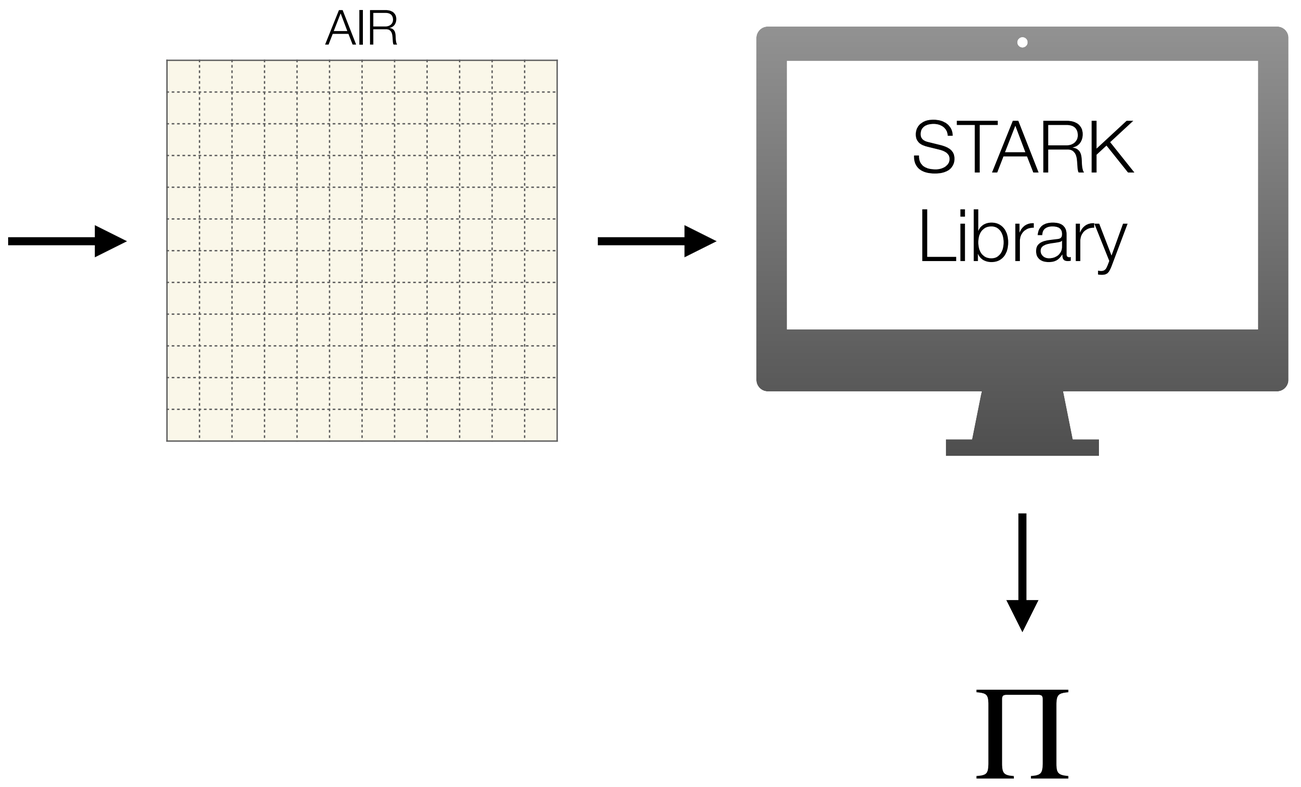
Pipeline of zkSTARKs

```rust
fn main() {
    // imagine complicated logic
    // ...
    // ...
    // ...

    println!("Hello, world!");
}
```



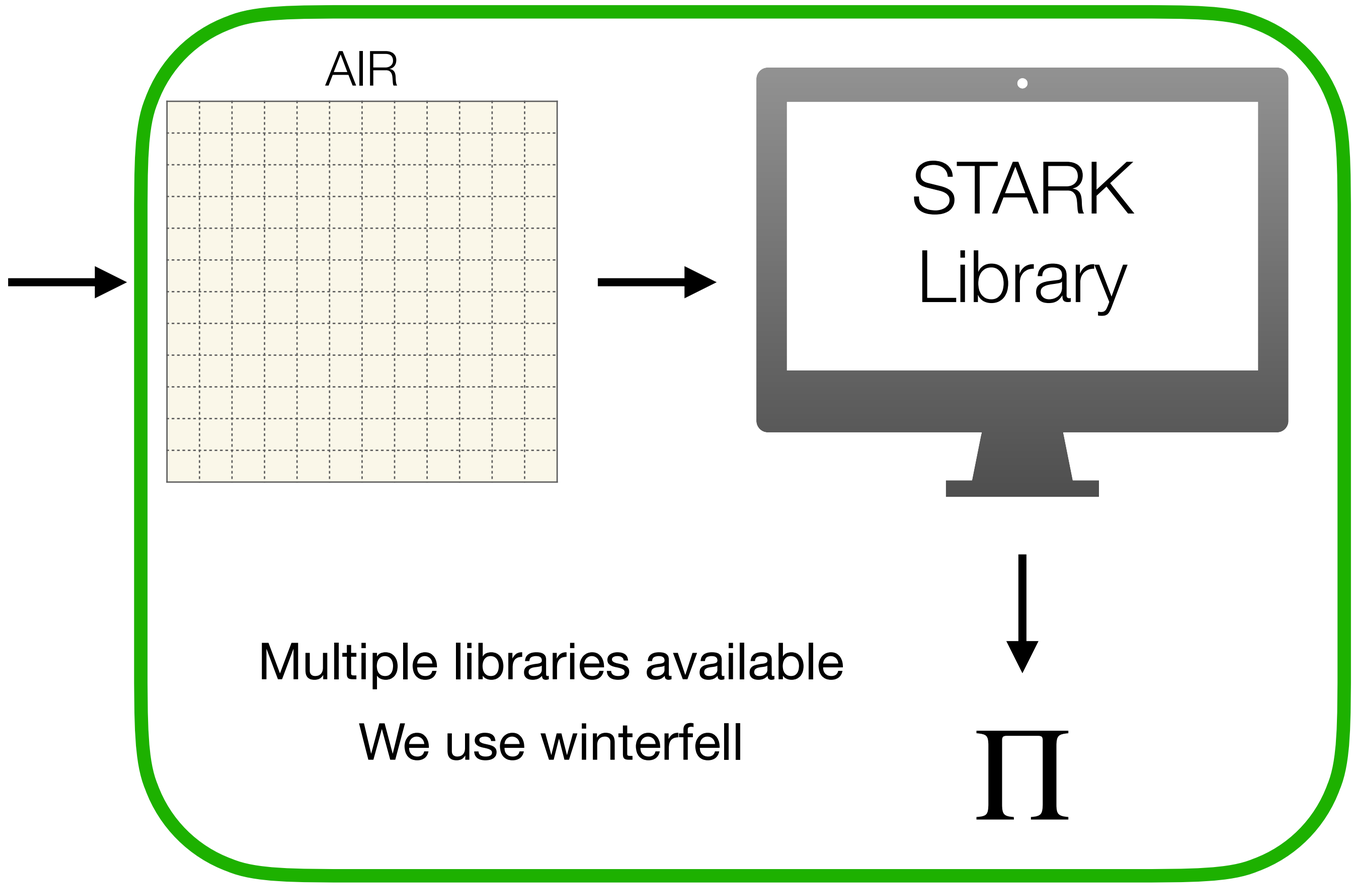
Pipeline of zkSTARKs

```
fn main() {
    // imagine complicated logic
    // ...
    // ...
    // ...

    println!("Hello, world!");
}
```



Pipeline of zkSTARKs

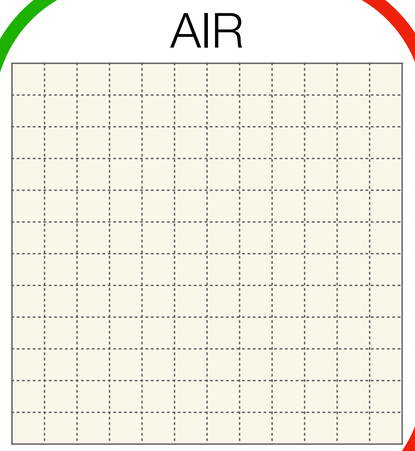
```rust
fn main() {
    // imagine complicated logic
    // ...
    // ...
    // ...

    println!("Hello, world!");
}
```



* *github.com/facebook/winterfell*

Pipeline of zkSTARKs

```
fn main() {
    // imagine complicated logic
    // ...
    // ...
    // ...

    println!("Hello, world!");
}
```

Π

Crucial for performance!

Needs careful hand optimization

Can reduce to “simpler” circuits

Multiple libraries available

We use winterfell

* *github.com/facebook/winterfell*

Dilithium Verification

Public Key: $(A, t) \in \mathcal{R}^{4\times4} \times \mathcal{R}^4$ **Signature:** $(z, \tilde{c}) \in \mathcal{R}^4 \times \{0,1\}^\lambda$

```
fn zkdilithium_verify() {
    // Fischer-Yates style
    c ← HashInBall(c̃)
    c ∈ {−1,0,1}^256, ||c||_1 = τ









}
```

Avoid rejection sampling

Dilithium Verification

Public Key: $(A, t) \in \mathcal{R}^{4\times4} \times \mathcal{R}^4$ **Signature:** $(z, \tilde{c}) \in \mathcal{R}^4 \times \{0,1\}^\lambda$

```
fn zkdilithium_verify() {
    // Fischer-Yates style
    c ← HashInBall(c̃)
    c ∈ {−1,0,1}^256, ||c||_1 = τ

    // Polynomial Multiplication
    w ← Az − ct









}
```

Avoid rejection sampling

Reduce to polynomial identity testing

Dilithium Verification

Public Key: $(A, t) \in \mathcal{R}^{4\times4} \times \mathcal{R}^4$ **Signature:** $(z, \tilde{c}) \in \mathcal{R}^4 \times \{0,1\}^\lambda$

```
fn zkdilithium_verify() {
    // Fischer-Yates style
    c ← HashInBall(c̃)
    c ∈ {−1,0,1}^256, ||c||_1 = τ

    // Polynomial Multiplication
    w ← Az − ct

    // Extract High bits of elements
    w_1 ← HighBits(w)









}
```

Avoid rejection sampling

Reduce to polynomial identity testing

Check decomposition instead of computing

Dilithium Verification

Public Key: $(A, t) \in \mathcal{R}^{4\times4} \times \mathcal{R}^4$ **Signature:** $(z, \tilde{c}) \in \mathcal{R}^4 \times \{0,1\}^\lambda$

```
fn zkdilithium_verify() {
    // Fischer-Yates style
    c ← HashInBall(c̃)
    c ∈ {−1,0,1}^256, ||c||_1 = τ

    // Polynomial Multiplication
    w ← Az − ct

    // Extract High bits of elements
    w_1 ← HighBits(w)

    // Hashing
    assert(c̃ == H(pk||msg||w_1))



}
```

Avoid rejection sampling

Reduce to polynomial identity testing

Check decomposition instead of computing

Use Poseidon hash [GKRRM21]

Dilithium Verification

Public Key: $(A, t) \in \mathcal{R}^{4\times4} \times \mathcal{R}^4$ **Signature:** $(z, \tilde{c}) \in \mathcal{R}^4 \times \{0,1\}^\lambda$

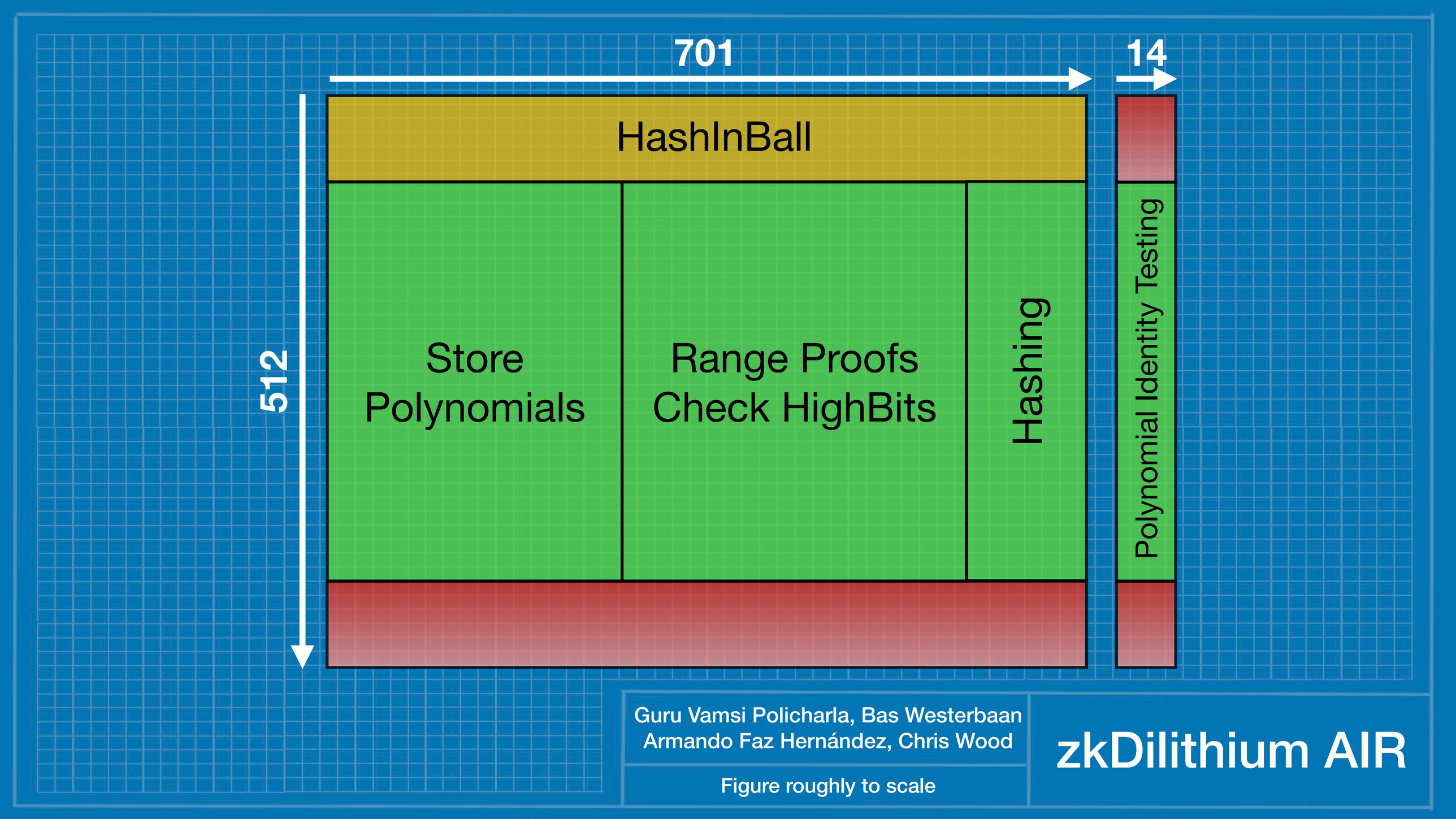
```
fn zkdilithium_verify() {
    // Fischer-Yates style
    c ← HashInBall(c̃)
    c ∈ {−1,0,1}^256, ||c||_1 = τ

    // Polynomial Multiplication
    w ← Az − ct

    // Extract High bits of elements
    w_1 ← HighBits(w)

    // Hashing
    assert(c̃ == H(pk||msg||w_1))

    // Range Proofs
    assert(||z||_∞ < …)
}
```

Avoid rejection sampling

Reduce to polynomial identity testing

Check decomposition instead of computing

Use Poseidon hash [GKRRM21]



Guru Vamsi Policharla, Bas Westerbaan
Armando Faz Hernández, Chris Wood

Figure roughly to scale

zkDilithium AIR

Try it yourself!

Cloudflare Research: Post-Quantum Privacy Pass

This website is a partial demo of the post-quantum anonymous credential scheme introduced in the paper titled Post-Quantum Privacy Pass via Post-Quantum Anonymous Credentials.

What is the demo?

The demo computes in the browser the proof of knowledge of a zkdilithium signature which acts as a blind signature. The proof is then sent to a Cloudflare Worker, which verifies it.

References

Paper: ia.cr/2023/414

Start

Log of Processing

⚙️ Client: starts generating proof
🏁 Client: proof completed
✈️ Server: send proof to server
➡️ Server response: Proof verification was successful.

Contact

You can reach us directly at ask-research@cloudflare.com with questions and feedback.

Cloudflare Research

zkdilithium.cloudflareresearch.com

github.com/guruvamsi-policharla/zkdilithium

Takeaways and Future work

- PQ Anonymous Credentials are *semi-practical!*
- *Careful tailoring of ZKPs to circuit being proved performs surprisingly well*
- *Design PQ signatures with proof verification in mind and vice versa?*
- *Formal verification for the AIR translation*
- *More details and new ideas for rate-limiting in the paper (eprint:2023/414)*

Thank you!

zkdilithium.cloudflareresearch.com