Foundations of High Speed Cryptography

Module 1 - Theory
Lesson 3 - Hashes and Merkle Trees

Hash Functions as random oracles

Hash functions as random oracles

- **Random oracle: A**n abstract, idealized function that, when given a unique query, returns a response chosen uniformly at random from its output domain.

  - **Unpredictability:** Given x and $H(x)$, it is infeasible to predict $H(x')$ for $x' \neq x$

  - **Inversion resistance:** Given $H(x)$ it is hard to invert H and get x

  - **Collision resistance:** It is computationally feasible to find any two distinct inputs $x \neq x' \mid H(x') = H(x)$

- Real world Collision resistant hash functions, approximate this behavior : **Pseudo random functions**

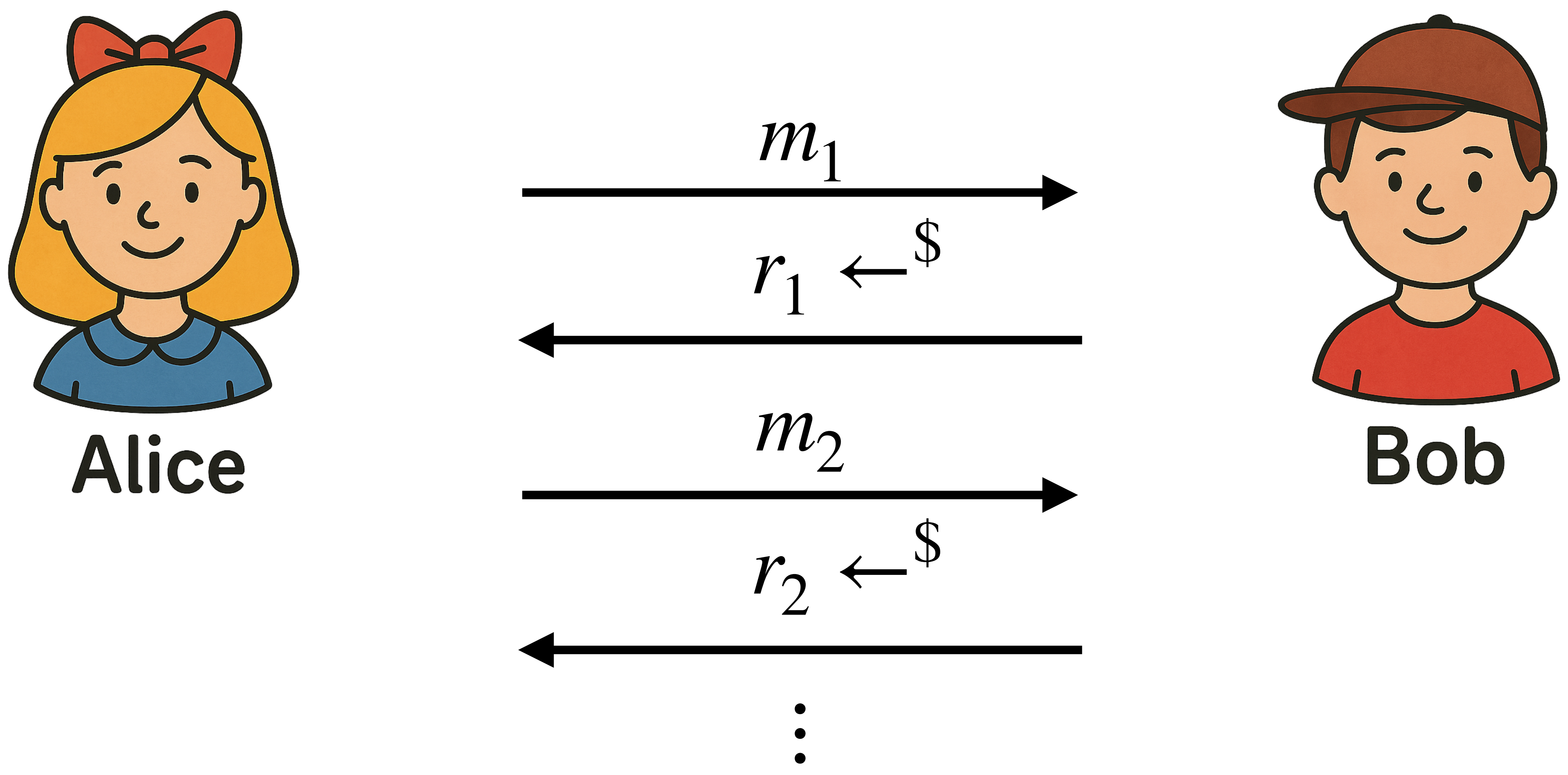
- A true random function would have a description that is exponentially large

Random oracles and Interactive Oracle Proofs

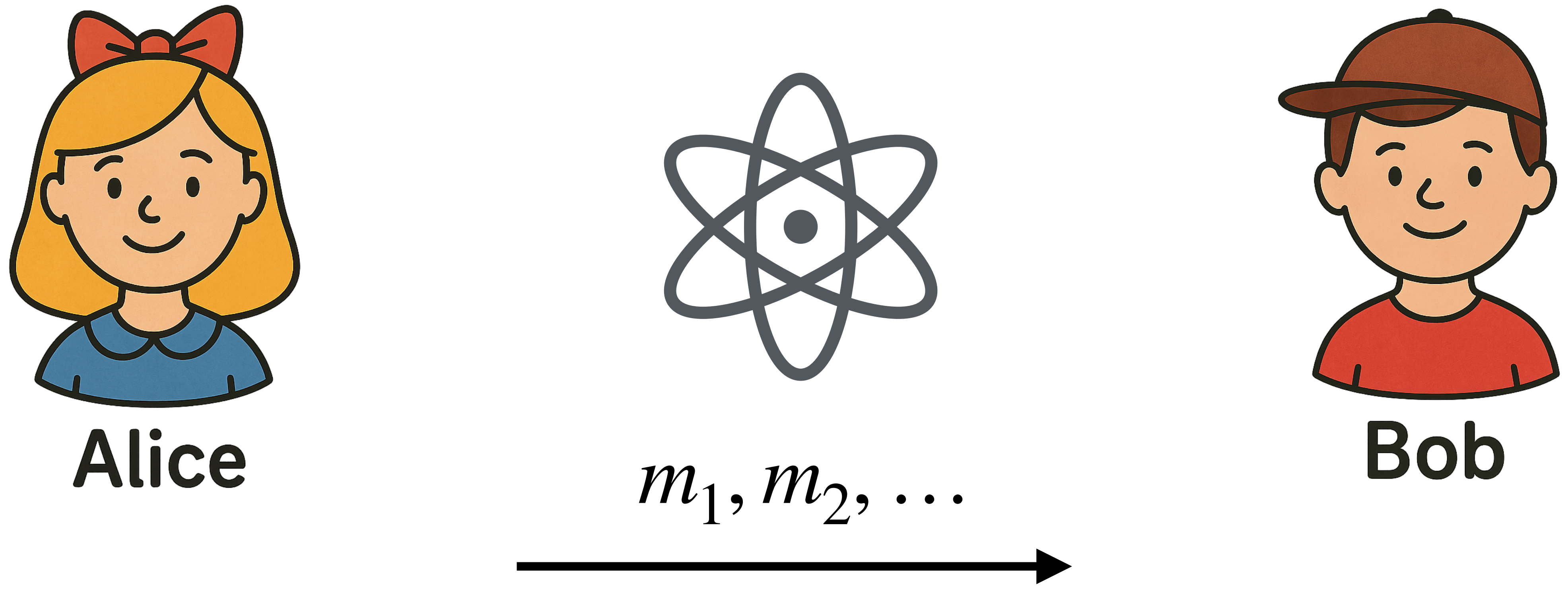
5.1 The Random Oracle Model

The random oracle model (ROM) [FS86, BR93] is an idealized setting meant to capture the fact that cryptographers have developed hash functions (e.g., SHA-3 or BLAKE3) that efficient algorithms seem totally unable to distinguish from random functions. By a random function R mapping some domain $\mathcal{D}$ to the κ-bit range $\{0,1\}^\kappa$, we mean the following: on any input $x \in \mathcal{D}$, R chooses its output $R(x)$ uniformly at random from $\{0,1\}^\kappa$.

Interactive protocol



Non-Interactive protocol



$$r_1 = R(x, m_1)$$
$$r_2 = R(x, r_1, m_2)$$
$$r_3 = R(x, r_2, m_3)$$
$$\vdots$$

https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.pdf

Fiat Shamir heuristic

- Fiat Shamir transformation maps an interactive argument to a non-interactive argument in a given oracle model (Random Oracle model)

$$
\begin{aligned}
r_1 &= R(x, m_1) \\
r_2 &= R(x, r_1, m_2) \\
r_3 &= R(x, r_2, m_3) \\
&\vdots
\end{aligned}
$$

- The verifier challenges in a multi round protocol is obtained by hash chaining the prover messages, with the output of the challenges in the prior round
    - Sequential binding between prover message/challenge across rounds
    - Uniqueness of transcript
    - Resistance to replay attacks
    - Non-Malleability
- All public values are included in the hash chain, and use collision resistant hashes

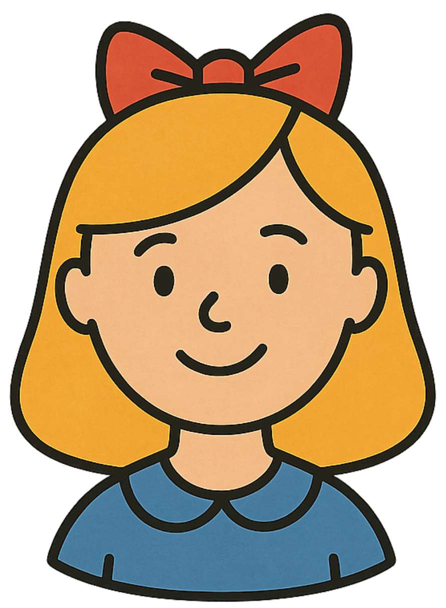
https://eprint.iacr.org/2025/536

https://www.zkdocs.com/docs/zkdocs/protocol-primitives/fiat-shamir/

Schnorr protocol : Interactive

- Alice wants to convince Bob that she knows a secret w, without revealing it.
- They agree to use the Schnorr Σ protocol for a DLOG relation

$$g \in \mathbb{G}$$



Alice 🖖

$$h = g^w \qquad \xrightarrow{\quad h \quad}$$

$$r \xleftarrow{\$} \mathbb{Z}_q^* $$

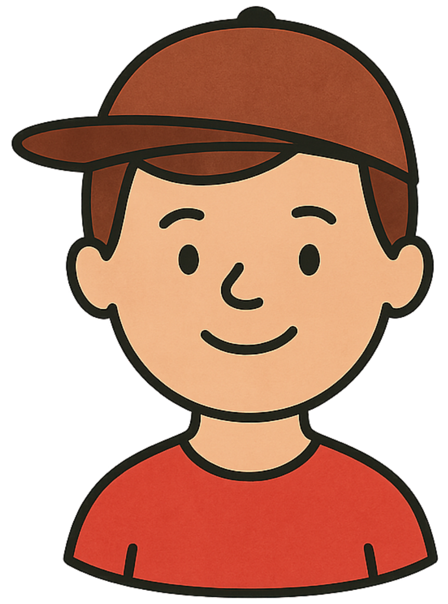
$$a = g^r \qquad \xrightarrow{\quad a \quad} \qquad a, h \neq 0, \quad a, h \in \mathbb{G}$$

$$\xleftarrow{\quad e \quad} \qquad e \xleftarrow{\$} \mathbb{Z}_q^*$$

$$z = w \cdot e + r \mod q \qquad \xrightarrow{\quad z \quad}$$

$$z \neq 0 \mod q$$

$$a \cdot h^e \stackrel{?}{=} g^z$$



Bob 🖖

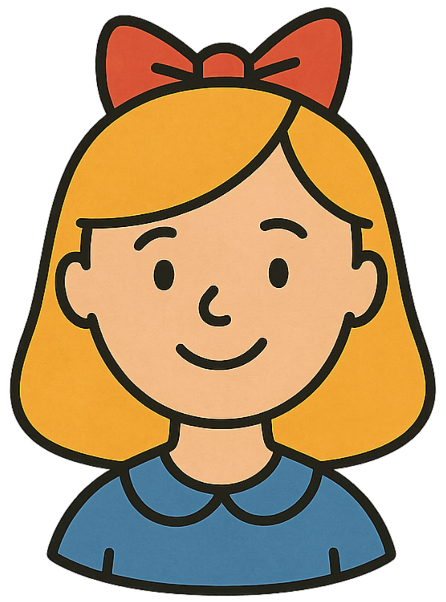
$$g^r \cdot h^e = g^r \cdot (g^w)^e = g^{r+w\cdot e} = g^z$$

https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.pdf

Schnorr protocol : Non-Interactive

- Alice wants to convince Bob that she knows a secret w, without revealing it.
- They agree to use the Schnorr Σ protocol for a DLOG relation

$$g \in \mathbb{G}$$



Alice

$$h = g^w$$

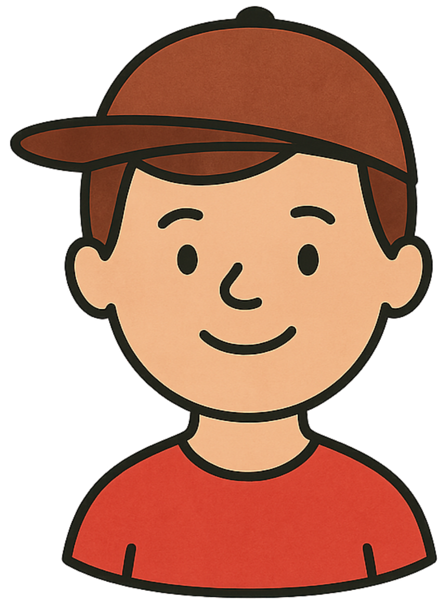
$$r \xleftarrow{\$} \mathbb{Z}_q^*$$

$$a = g^r$$

$$e = Hash(g, h, q, a)$$

$$z = w \cdot e + r \mod q$$

$$\xrightarrow{z, a, h}$$



Bob

$$a, h \neq 0$$

$$a, h \in \mathbb{G}$$

$$z \neq 0 \mod q$$

$$e = Hash(g, h, q, a)$$

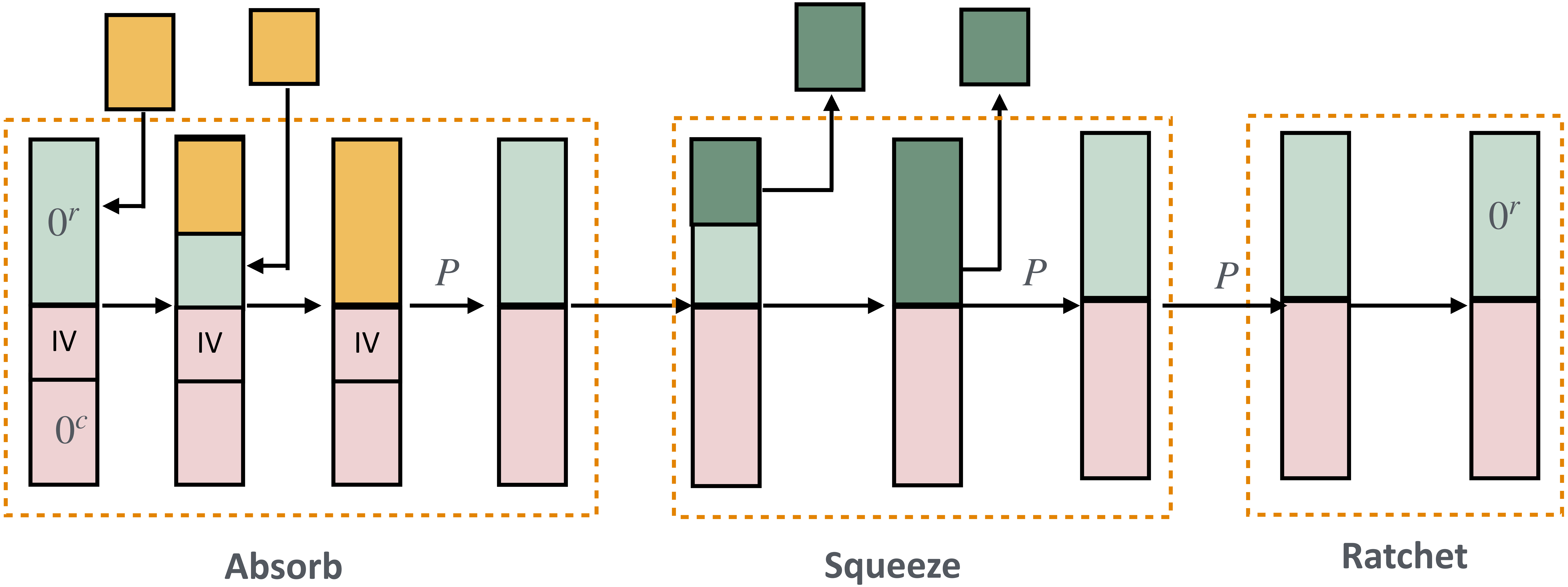
$$a \cdot h^e \stackrel{?}{=} g^z$$

$$g^r \cdot h^e = g^r \cdot (g^w)^e = g^{r+w\cdot e} = g^z$$

https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.pdf

Sponges for Fiat-Shamir

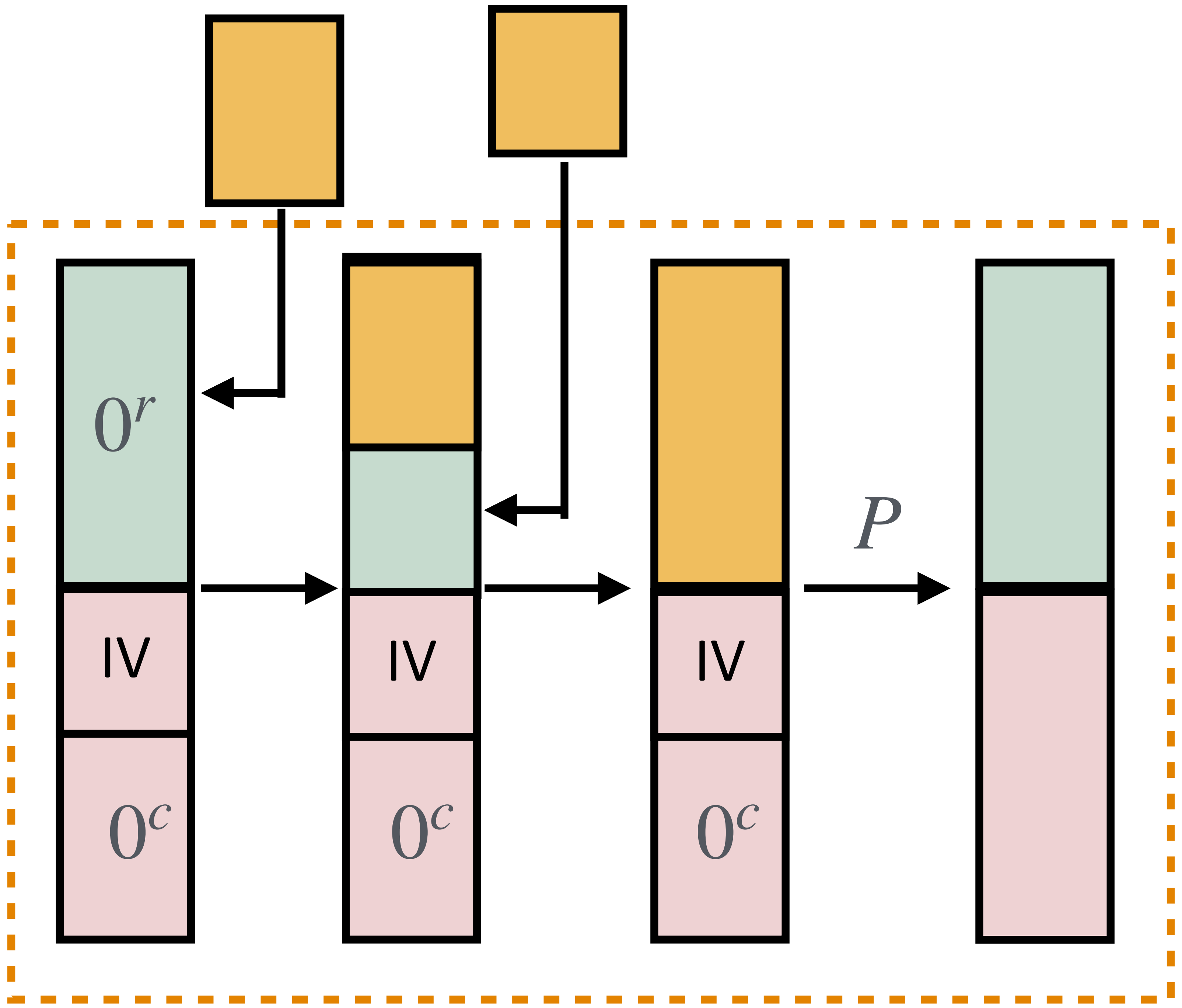
- Spongefish uses **stateful sponge** hashes in overwrite mode for Fiat-Shamir transformation



- A read/write "rate part" and an internal state "Capacity" part private to the hash.
- Hash chaining is maintained by the internal state of the sponge, no need for multiple hash invocations.

https://github.com/arkworks-rs/spongefish https://eprint.iacr.org/2025/536

Sponges for Fiat-Shamir - Absorb



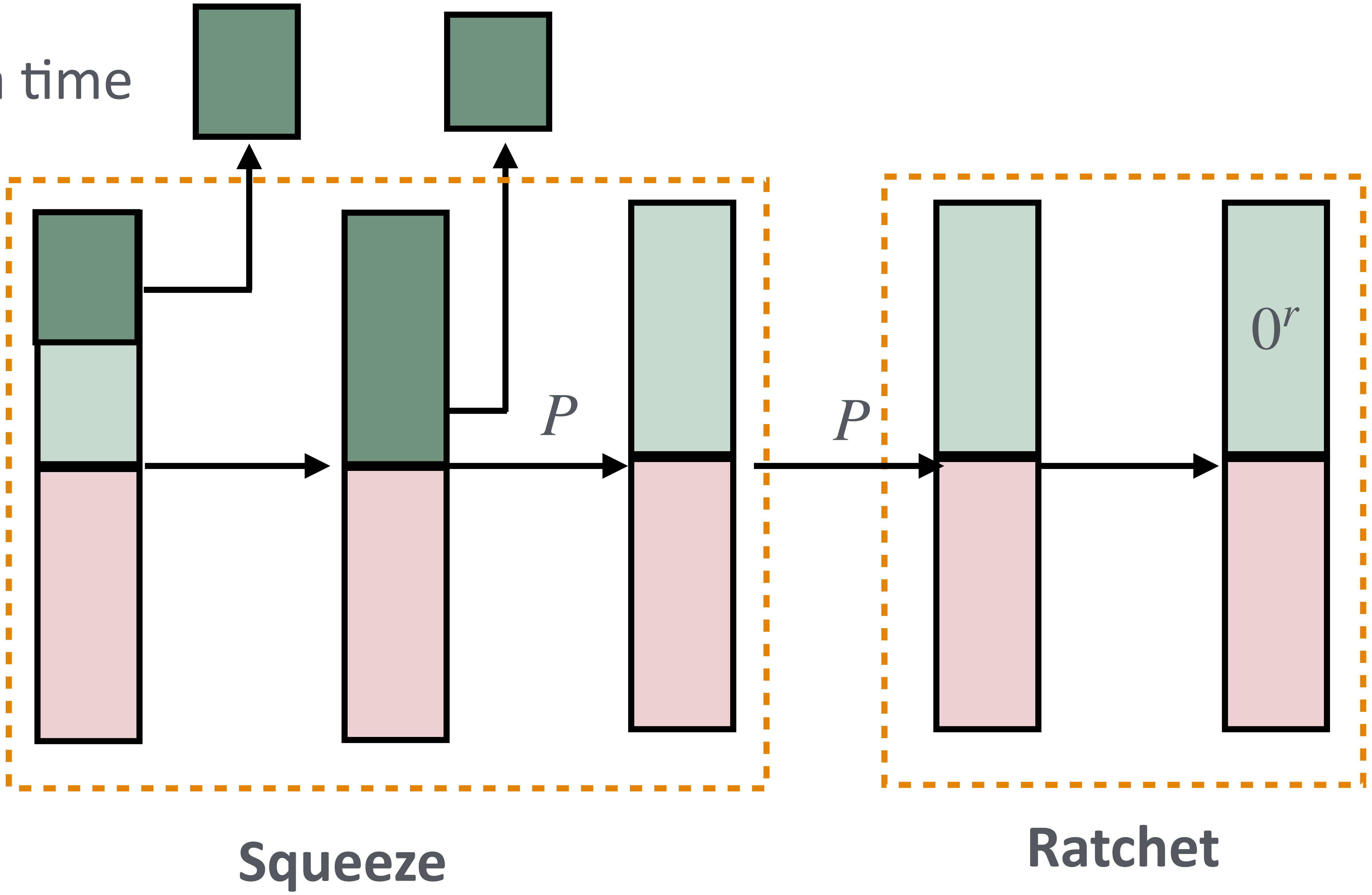
Absorb

- Absorb can **write to the rate part** upto "r" bytes at a time.
- Each time the rate part is saturated, a permute operation is triggered.
- All meta data such as labels, are encoded in the Initialization Vector in the instantiation.
- The only data absorbed by the sponge is the data generated by the prover. (Good for on-device proving)
- In overwrite mode, existing data in the state is overwritten rather than XOR or Field operations - Efficient

https://eprint.iacr.org/2025/536
https://github.com/arkworks-rs/spongefish

Sponges for Fiat-Shamir - Squeeze and Ratchet

- Internal state “capacity” is private to the sponge
- **Read from rate part** upto “r” bytes at a time
- Saturating read bytes from rate triggers a permute operation
- Makes it easy to generate multiple challenges if needed
- Ratchet allows clearing of state for protocols that need forward secrecy/state erasure



https://eprint.iacr.org/2025/536
https://github.com/arkworks-rs/spongefish

Domain separator as a formatted string

- Encode prover sequence of absorb/ squeeze as as formatted domain separator string

$$h = g^w$$

$$r \leftarrow^{\$} \mathbb{Z}_q^*$$

$$a = g^r$$

$$e = Hash(g, h, q, a)$$

$$z = w \cdot e + r \mod |\mathbb{G}|$$

- r: random nonce is not generated from FS sponge!, but rather from a secure CSPRNG.

```
fn add_schnorr_domain_separator<G: Group, H: SpongeInterface<u8>>(
) -> DomainSeparator<H, u8>
where DomainSeparator<H, u8>: GroupDomainSeparator<G> +
FieldDomainSeparator<G::ScalarField>,
{
   DomainSeparator::new("spongefish")
      .add_points(1, “gen")                          // generator :g
      .add_points(1, “pk”)                           // public key : h
     .add_scalars(1.”q”)                             //group order
      .ratchet()
      .add_points(1, “comm”)                         //commitment : a
      .challenge_scalars(1, “e”)                     //challenge : e
      .add_scalars(1, “resp")                        // response : z
}
let domain_separator =
   add_schnorr_domain_separator::<curve,hash>();

assert_eq!(
   domain_separator.as_bytes(),
   b”spongefish\0A32gen\0A32pk\0A32q\0R\0A32comm\0S32e\0A32resp”
);
```

https://github.com/arkworks-rs/spongefish

Domain separator as a formatted string

- Given domain separator string

b"**spongefish**\0**A**32**gen**\0**A**32**pk**\0**A**32**q**\0**R**\0**A**32**comm**\0**S**32**e**\0**A**32**resp**"

- Sponge state is initialized by the IV generated from the domain seperator

$$[0^r || IV_{32} || 0^{n-r-32}]$$

- Sponge executes **only the given sequence of Opcodes** generated from the formatted string

Ops = vec![**Absorb(**32)**,Absorb(**32), **Absorb(**32)**,Ratchet(),Absorb(**32)**,Squeeze(**32)**,Absorb(**32)**]**

- Stateful: absorb/squeeze/ratchet only affects the rate part, and the capacity part is private, and represents internal state of the sponge.

- Setting IV from the formatted string ensures both prover/verifier start from the same state.

https://eprint.iacr.org/2025/536
https://github.com/arkworks-rs/spongefish

Spongefish summary

- Use Collision resistant hash functions such as SHA256/Keccak/Blake3/ Poseidon2 etc
- Domain separator:
  - Unique, Deterministic protocol description
  - No Null bytes/integers in labels
  - Public data included in Domain separator definition
- Transcript consistency
  - Operator order enforcement
  - Transcript binding: included all prover messages in absorb
- ZK and randomness
  - Prover private CSPRNG (can be a private sponge)
  - State erasure at critical junctures (Ratchet)
  - No clone/copy prover state

https://eprint.iacr.org/2025/536

References

- **Cryptographic Hash Functions (CHF)**

  https://homes.esat.kuleuven.be/~preneel/phd_preneel_feb1993.pdf
- **SHA**

  https://chemejon.wordpress.com/2021/12/06/sha-3-explained-in-plain-english/

  https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf
- **Sponges (SAFE API)**

  https://eprint.iacr.org/2023/522
- **Merkle Tree**

  https://snargsbook.org/
- **Fiat-Shamir**

  https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.pdf

  https://eprint.iacr.org/2025/536

  https://github.com/arkworks-rs/spongefish
- **Examples/exercises**

  https://github.com/ingonyama-zk/research_POCs/tree/course/course