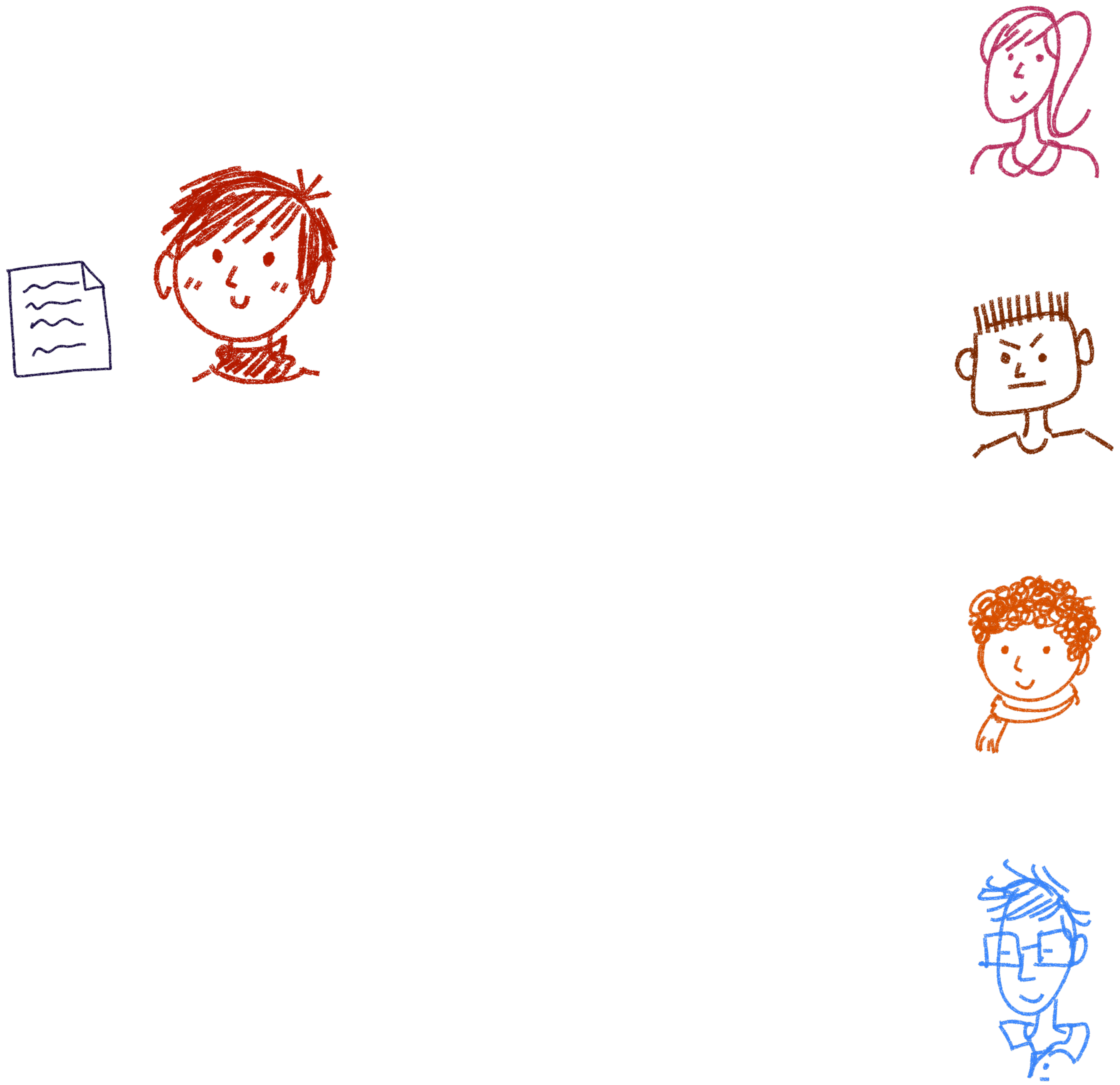
An Unstoppable Ideal Functionality for Signatures and a Modular Analysis of the Dolev-Strong Broadcast

Ran Cohen, Jack Doerner, Eysa Lee, Anna Lysyanskaya, and La(wre)nce Roy

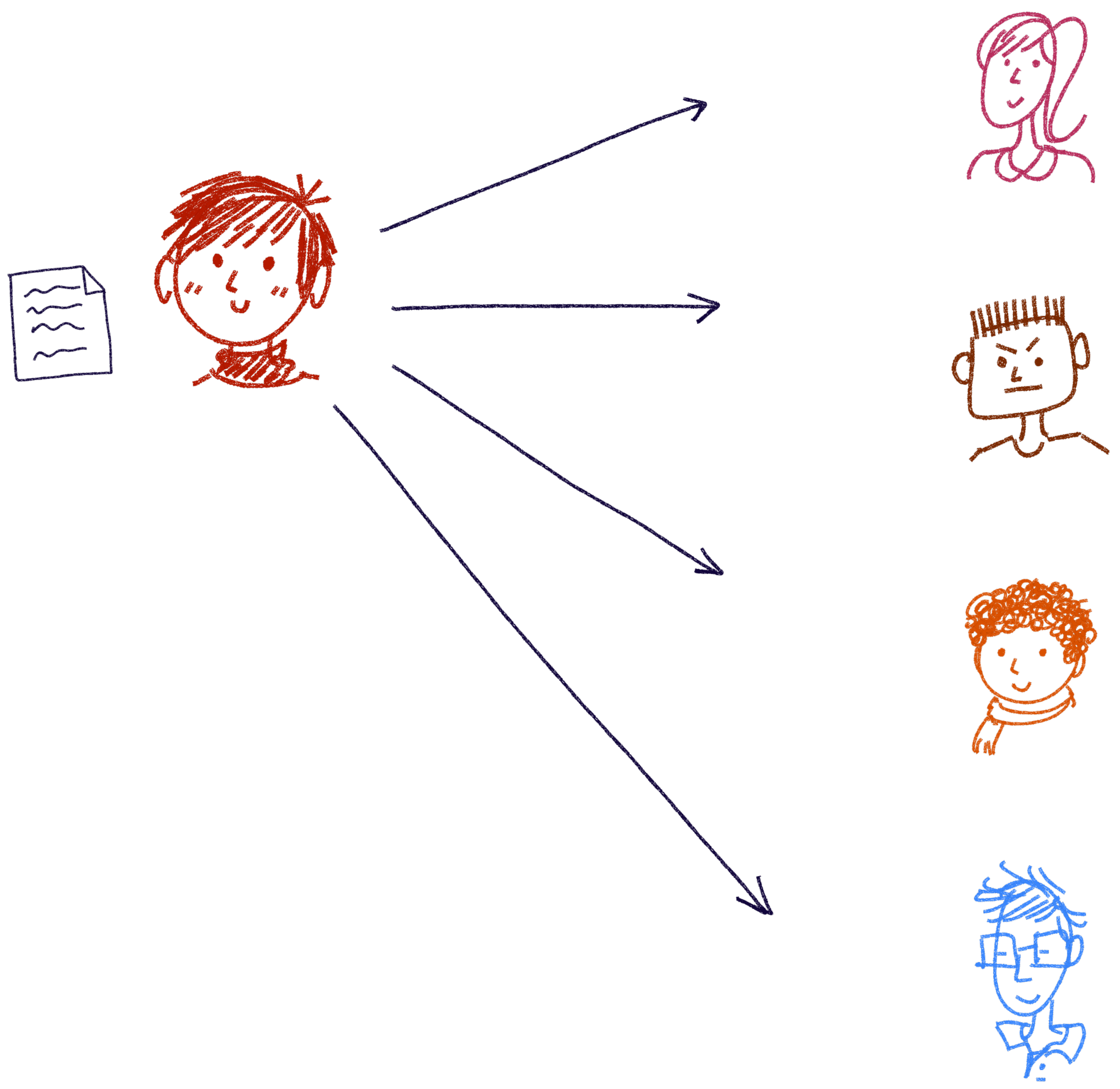
Reichman University · University of Virginia · Barnard College · Brown · Aarhus University

ePrint/2024/1807

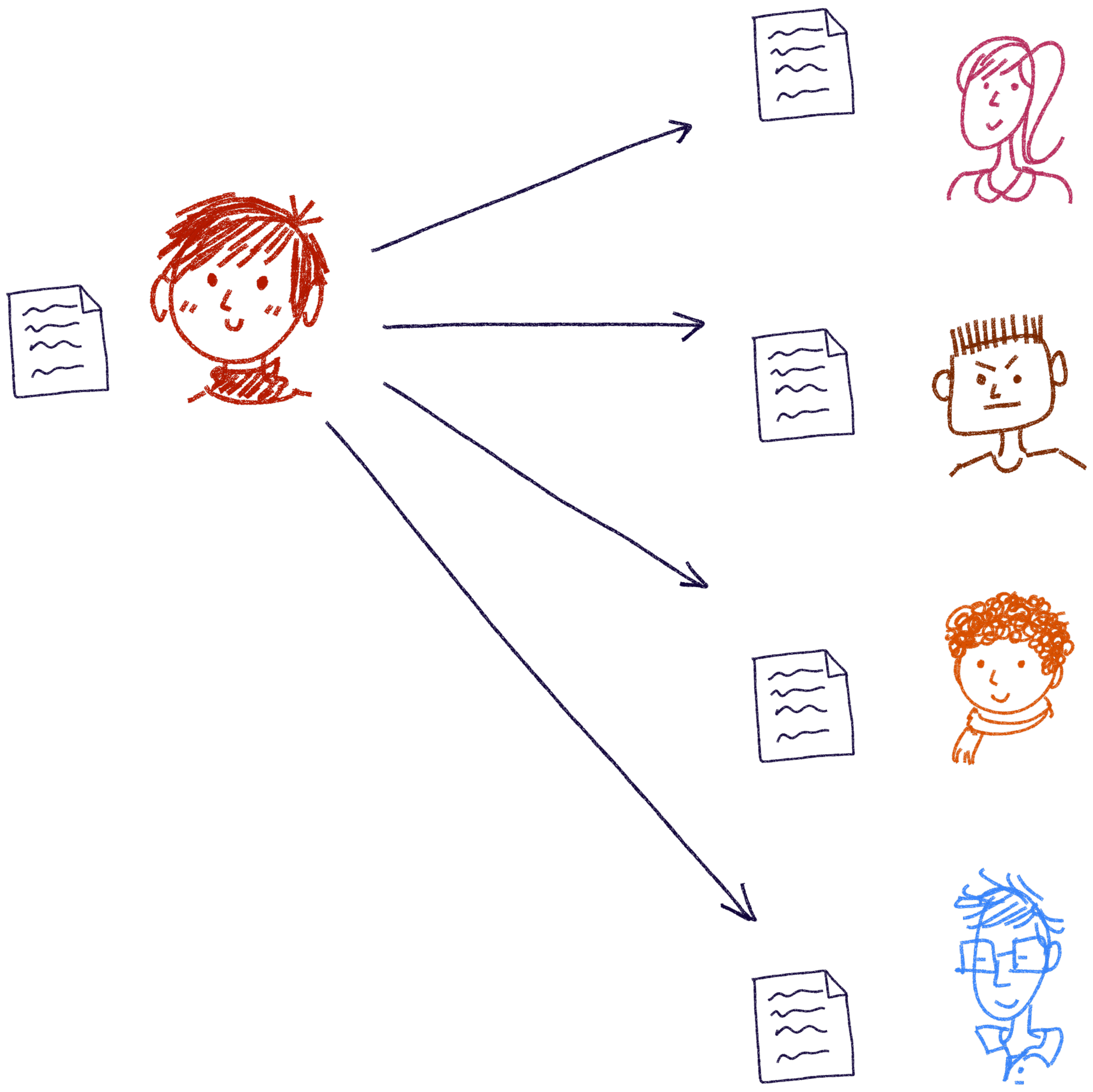
Motivating Example: Broadcast



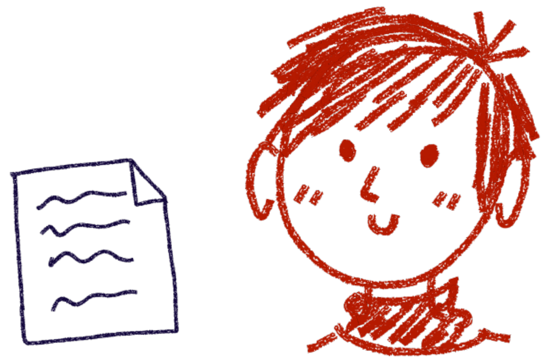
Motivating Example: Broadcast



Motivating Example: Broadcast

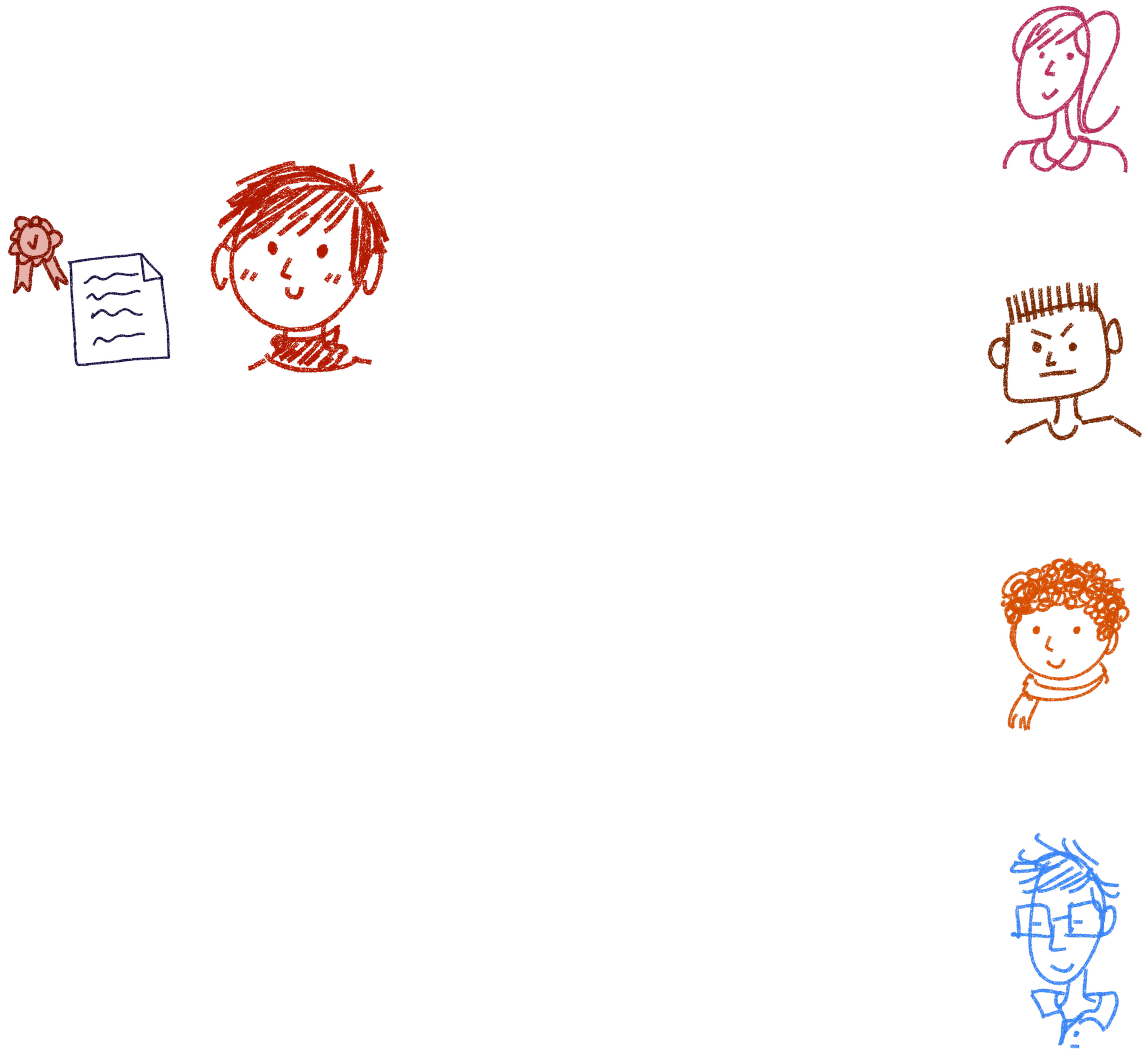


Dolev-Strong Broadcast [DS83]



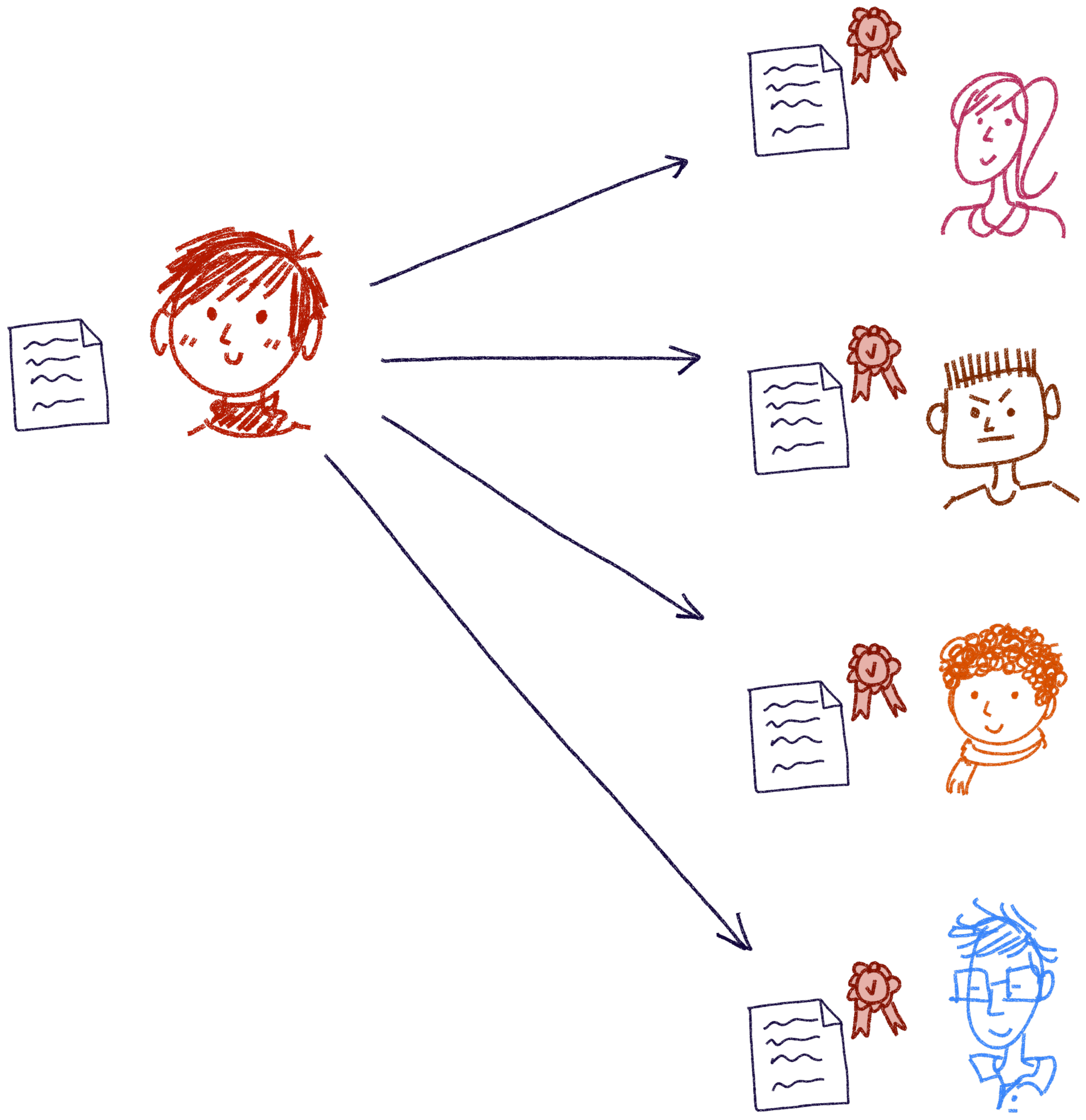
Dolev-Strong Broadcast [DS83]

Round 1: Sender signs a message and multicasts



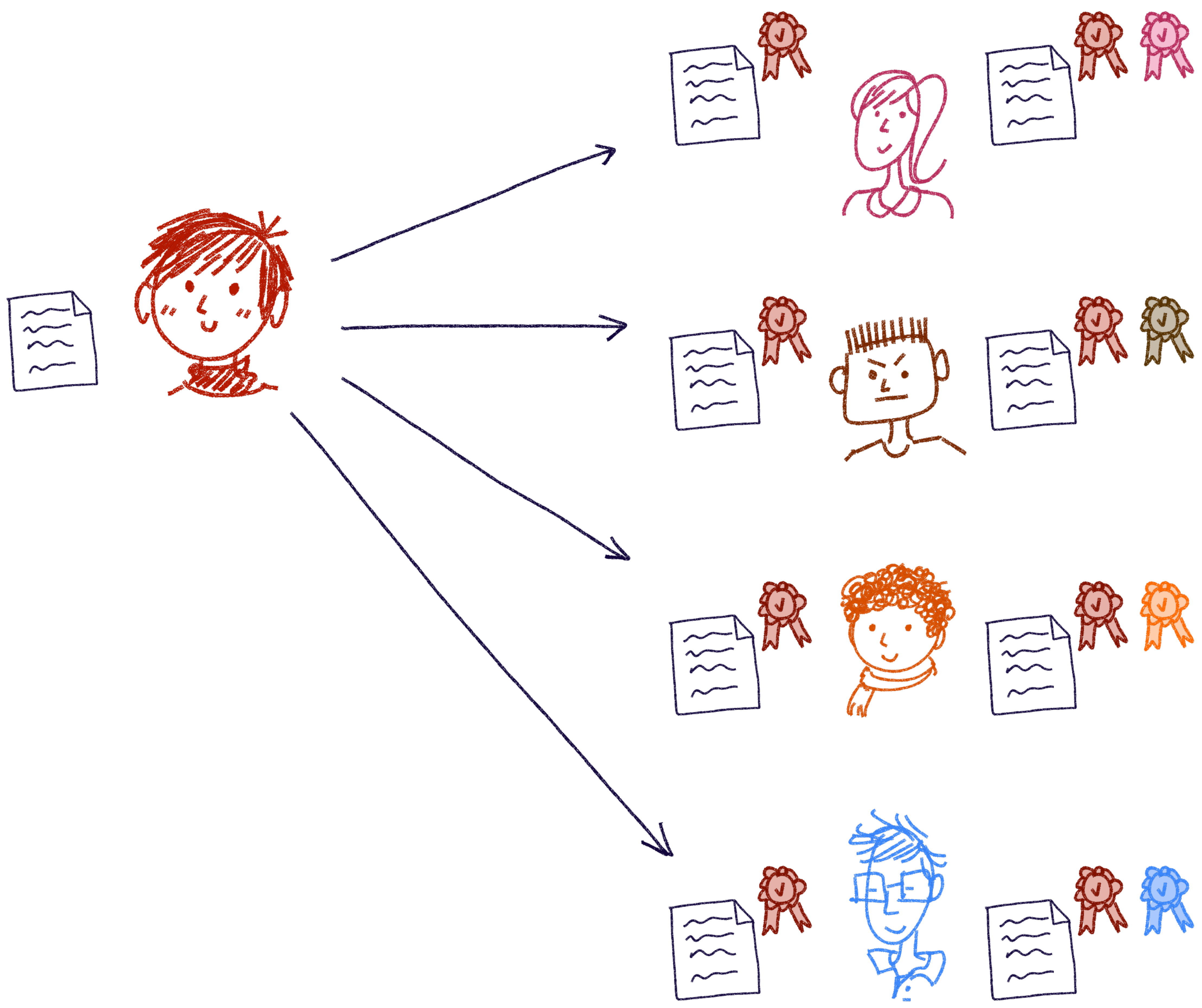
Dolev-Strong Broadcast [DS83]

Round 1: Sender signs a message and multicasts



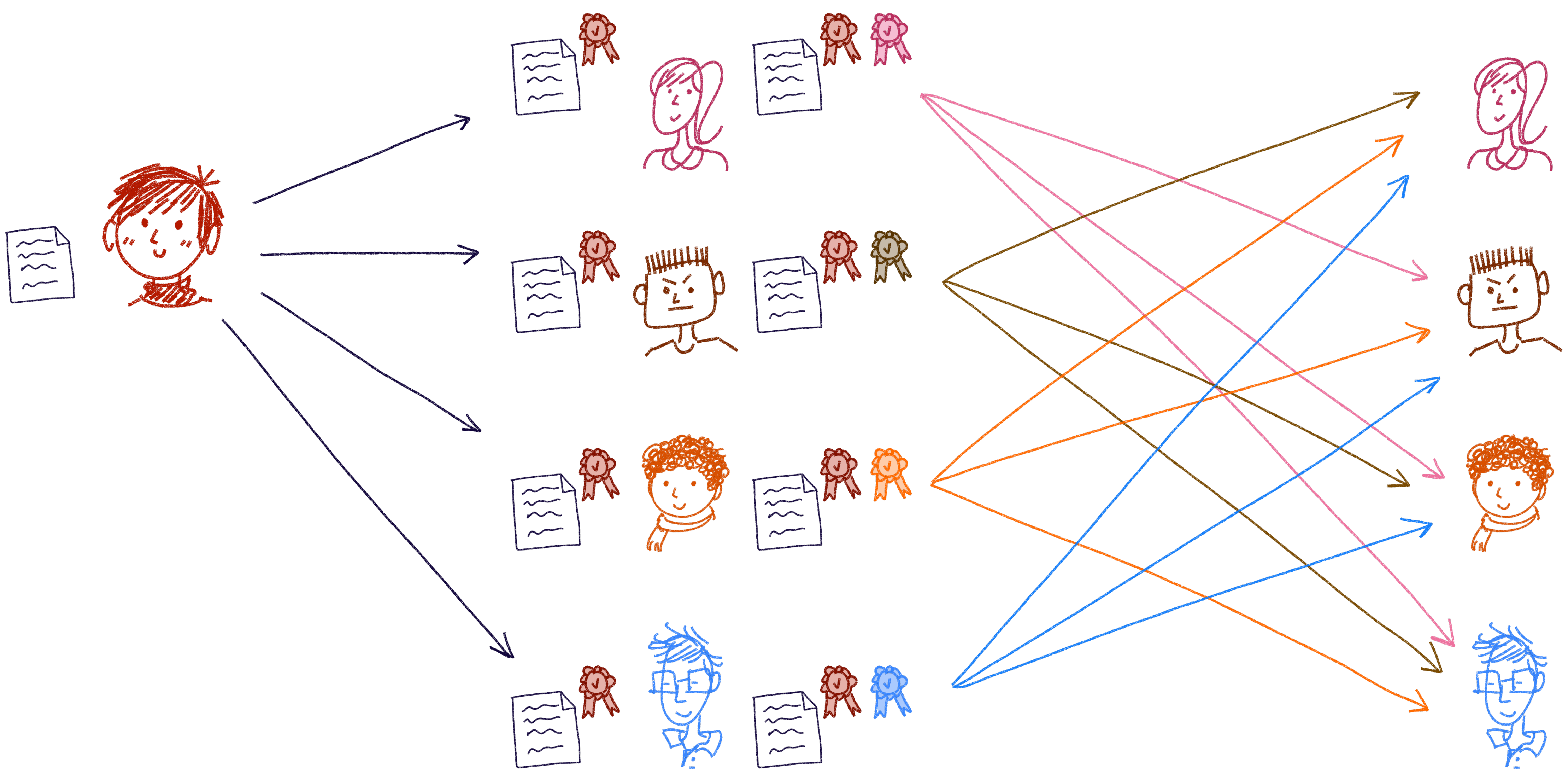
Dolev-Strong Broadcast [DS83]

Round 1: Sender signs a message and multicasts
Round r: If incoming message is “valid” and “new”, sign and multicast



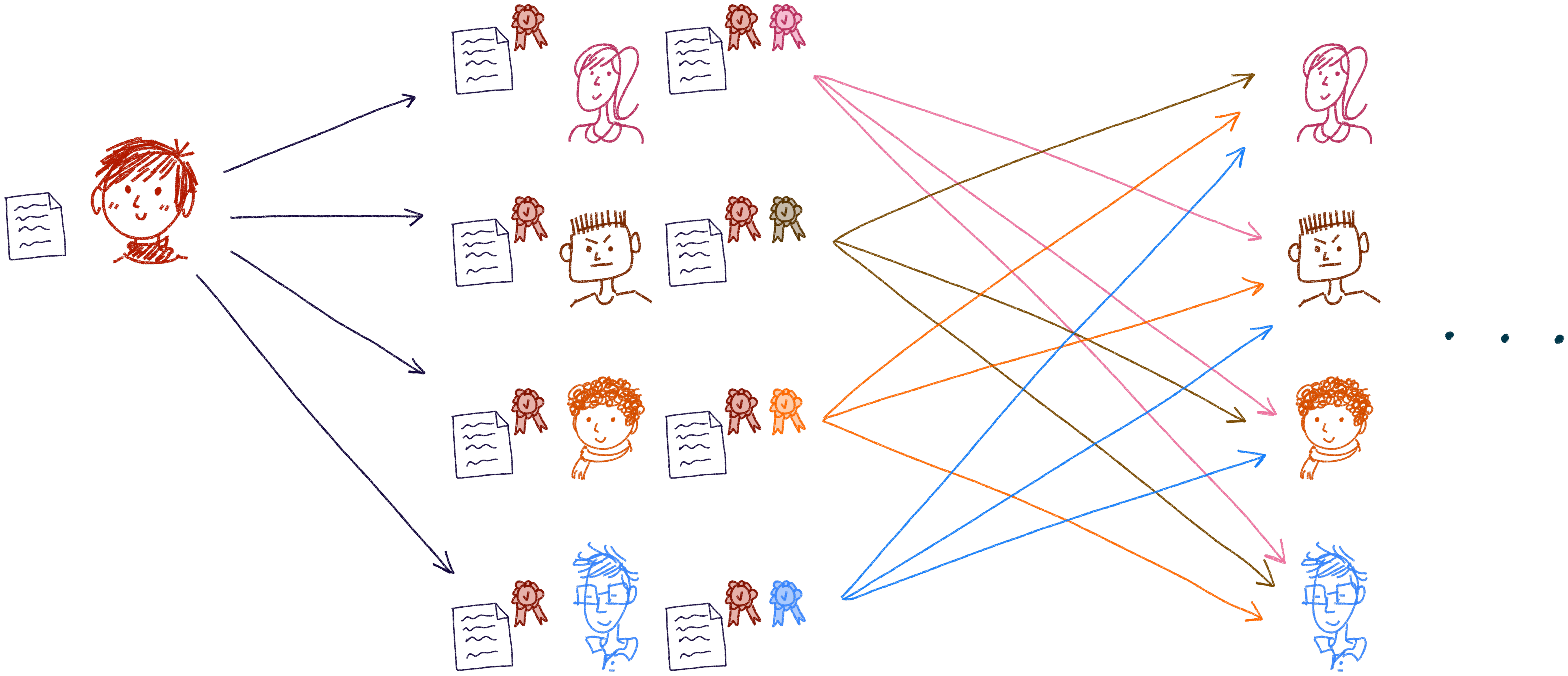
Dolev-Strong Broadcast [DS83]

Round 1: Sender signs a message and multicasts
Round r: If incoming message is “valid” and “new”, sign and multicast

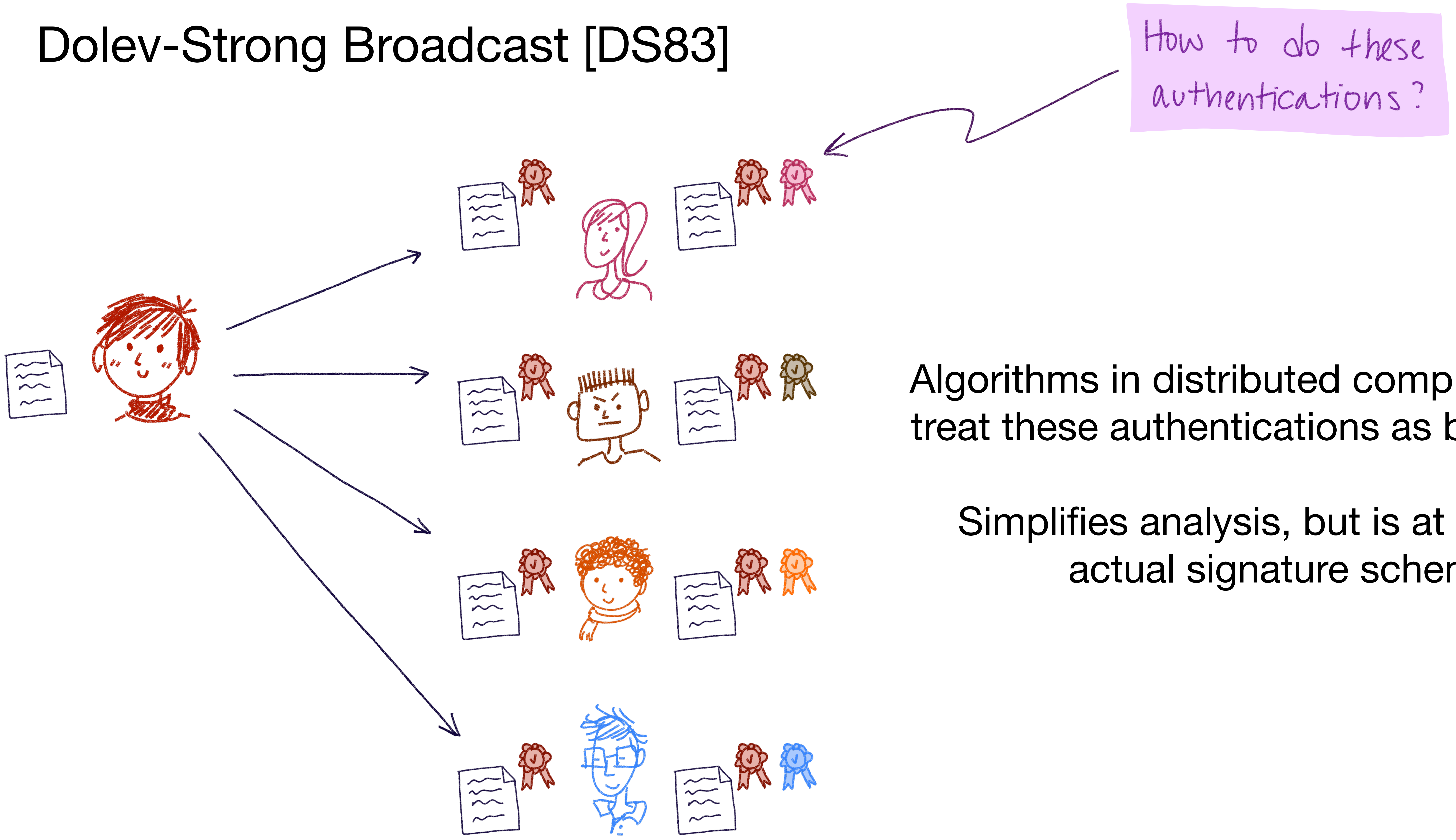


Dolev-Strong Broadcast [DS83]

Round 1: Sender signs a message and multicasts
Round r: If incoming message is "valid" and "new", sign and multicast
Round t+1: If you only received one "valid" message, output message.
Otherwise output bot



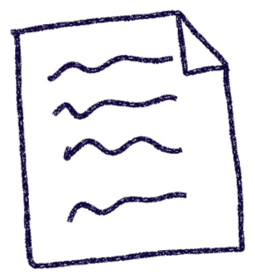
Dolev-Strong Broadcast [DS83]



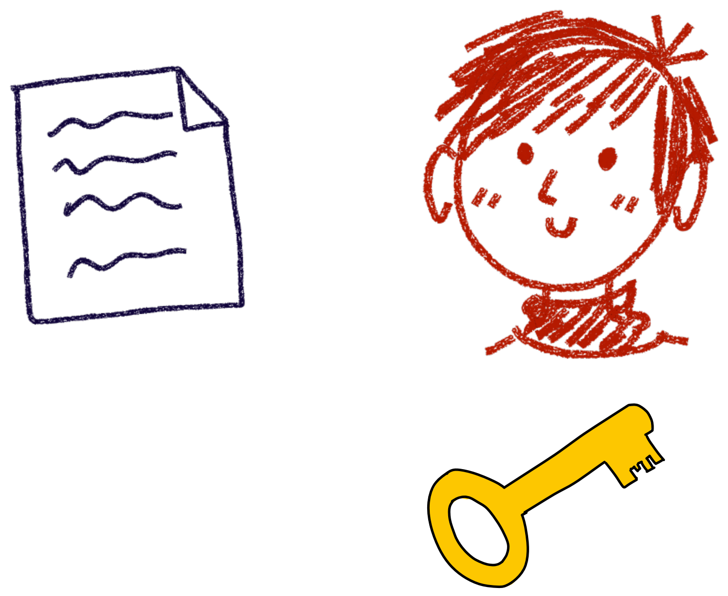
Algorithms in distributed computing tend to treat these authentications as being *perfect*

Simplifies analysis, but is at odds with actual signature schemes

Digital Signatures



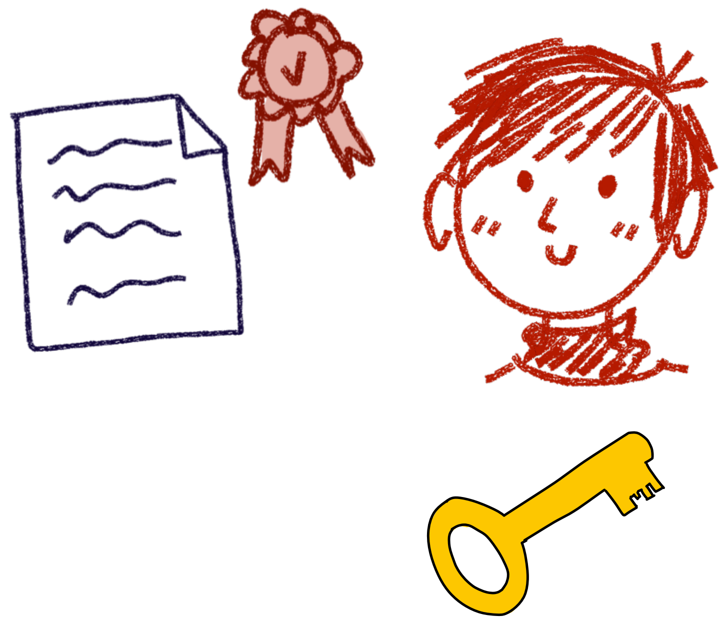
Digital Signatures



Signature Algorithms:

- KeyGen

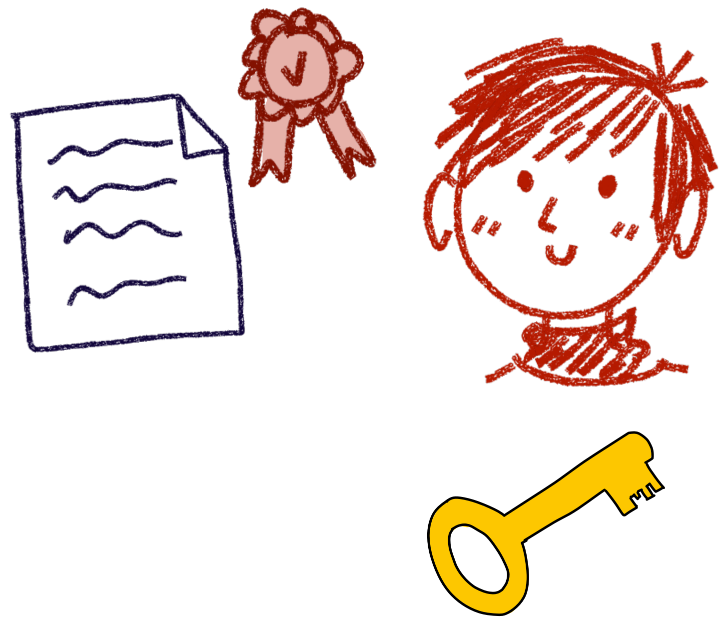
Digital Signatures



Signature Algorithms:

- KeyGen
- Sign

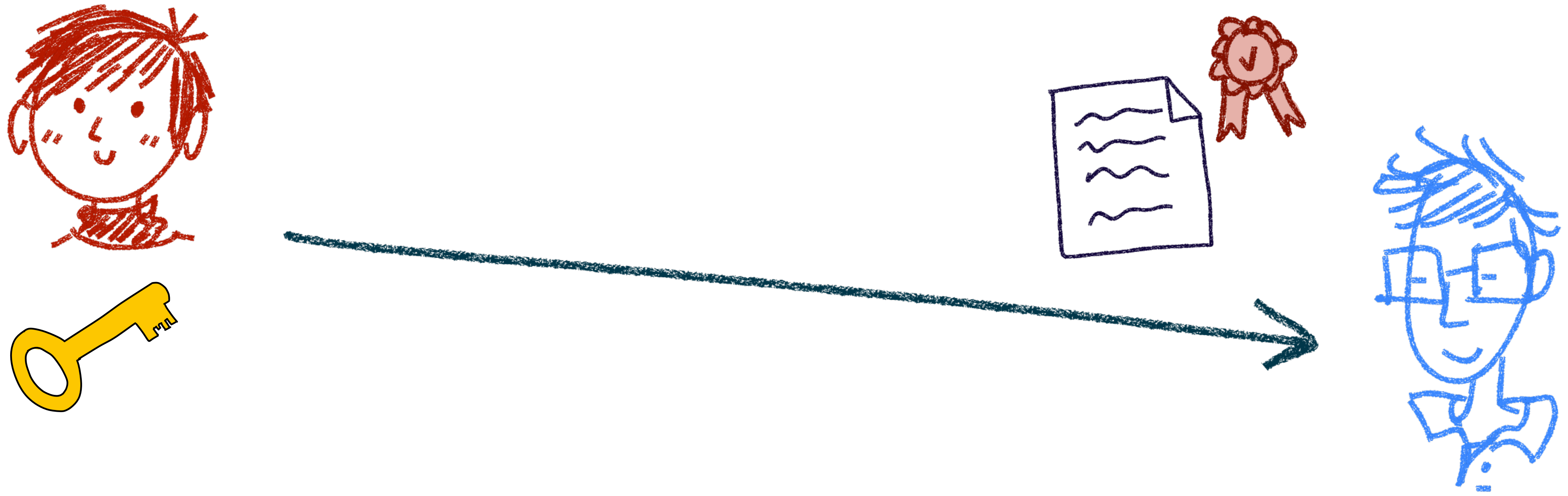
Digital Signatures



Signature Algorithms:

- KeyGen
- Sign
- Verify

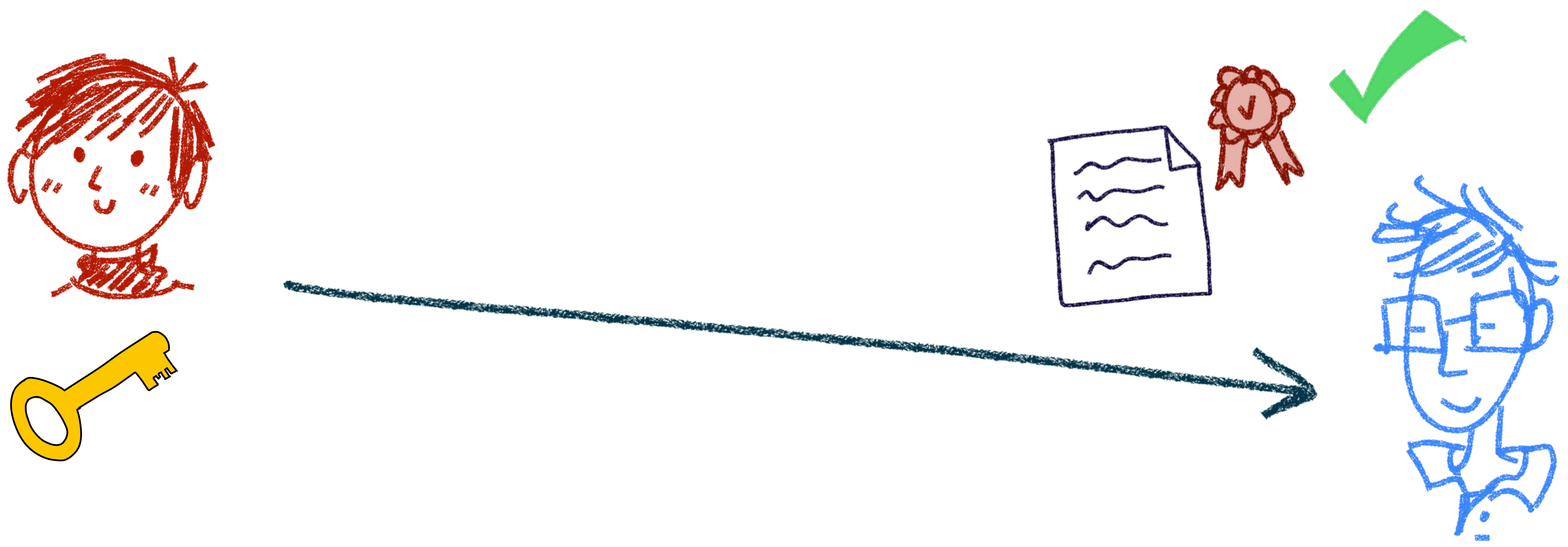
Digital Signatures



Signature Algorithms:

- KeyGen
- Sign
- Verify

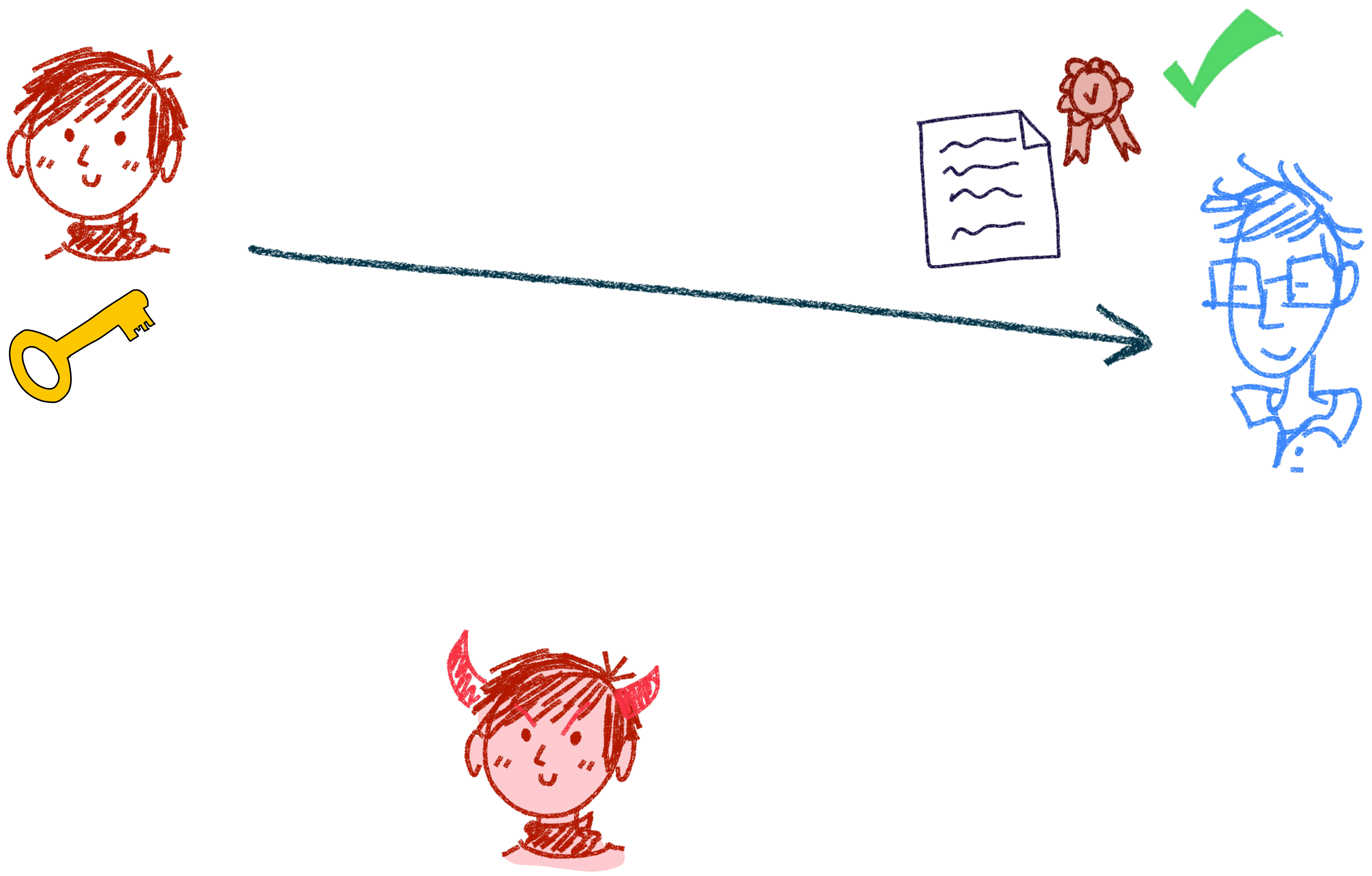
Digital Signatures



Signature Algorithms:

- KeyGen
- Sign
- Verify

Digital Signatures



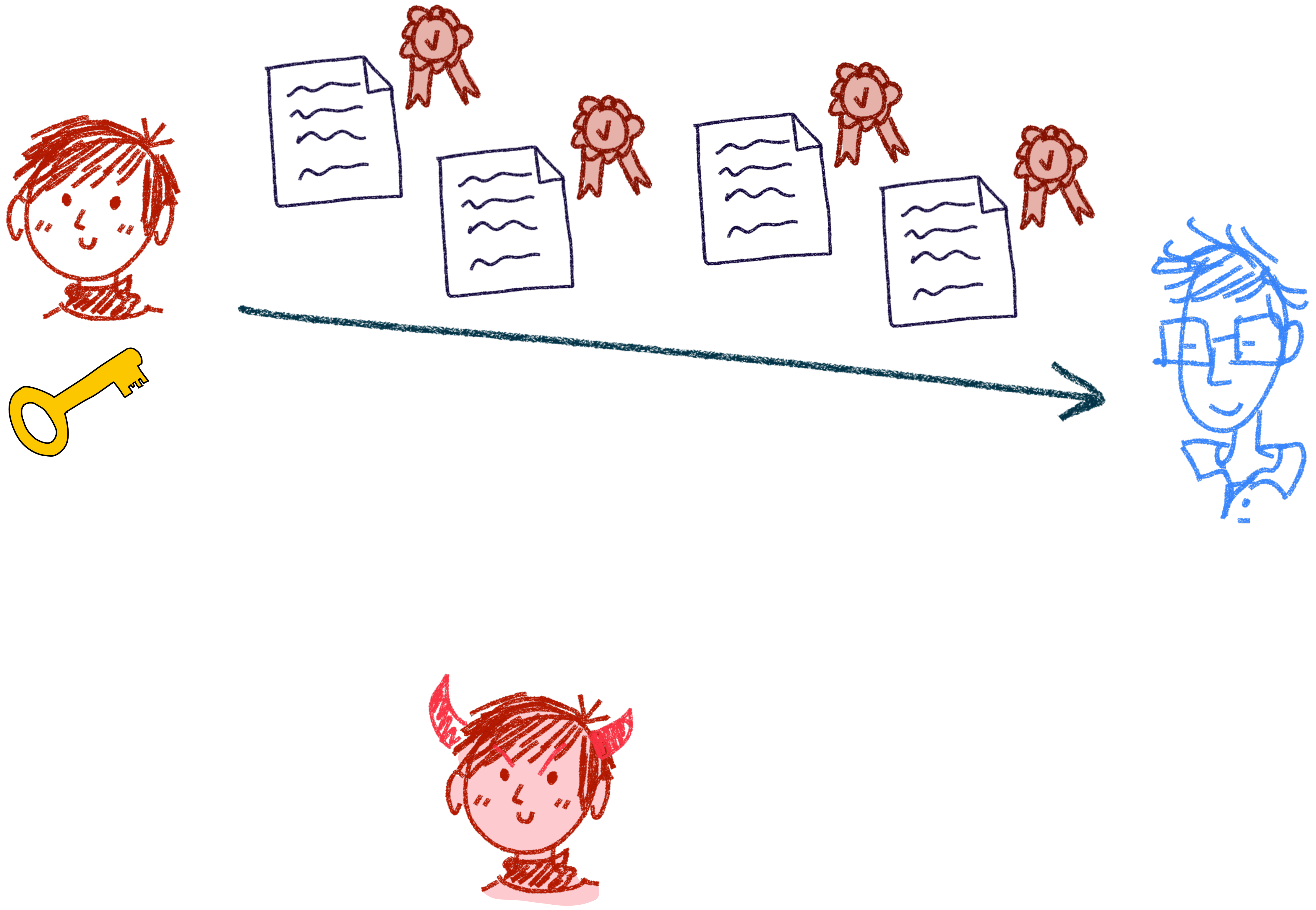
Signature Algorithms:

- KeyGen
- Sign
- Verify

Properties:

- Correctness
- Existential Unforgeability (EUF-CMA)

Digital Signatures



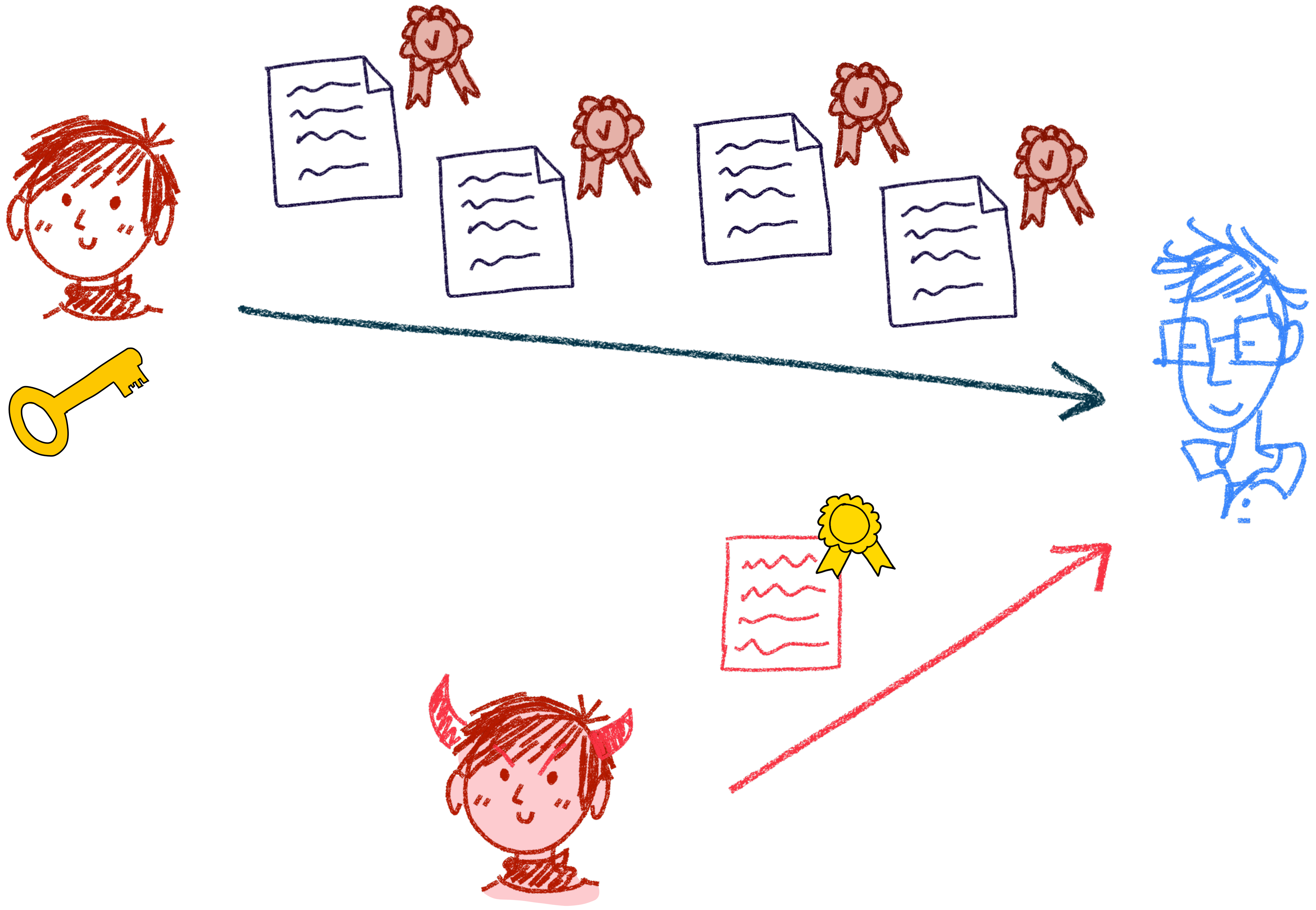
Signature Algorithms:

- KeyGen
- Sign
- Verify

Properties:

- Correctness
- Existential Unforgeability (EUF-CMA)

Digital Signatures



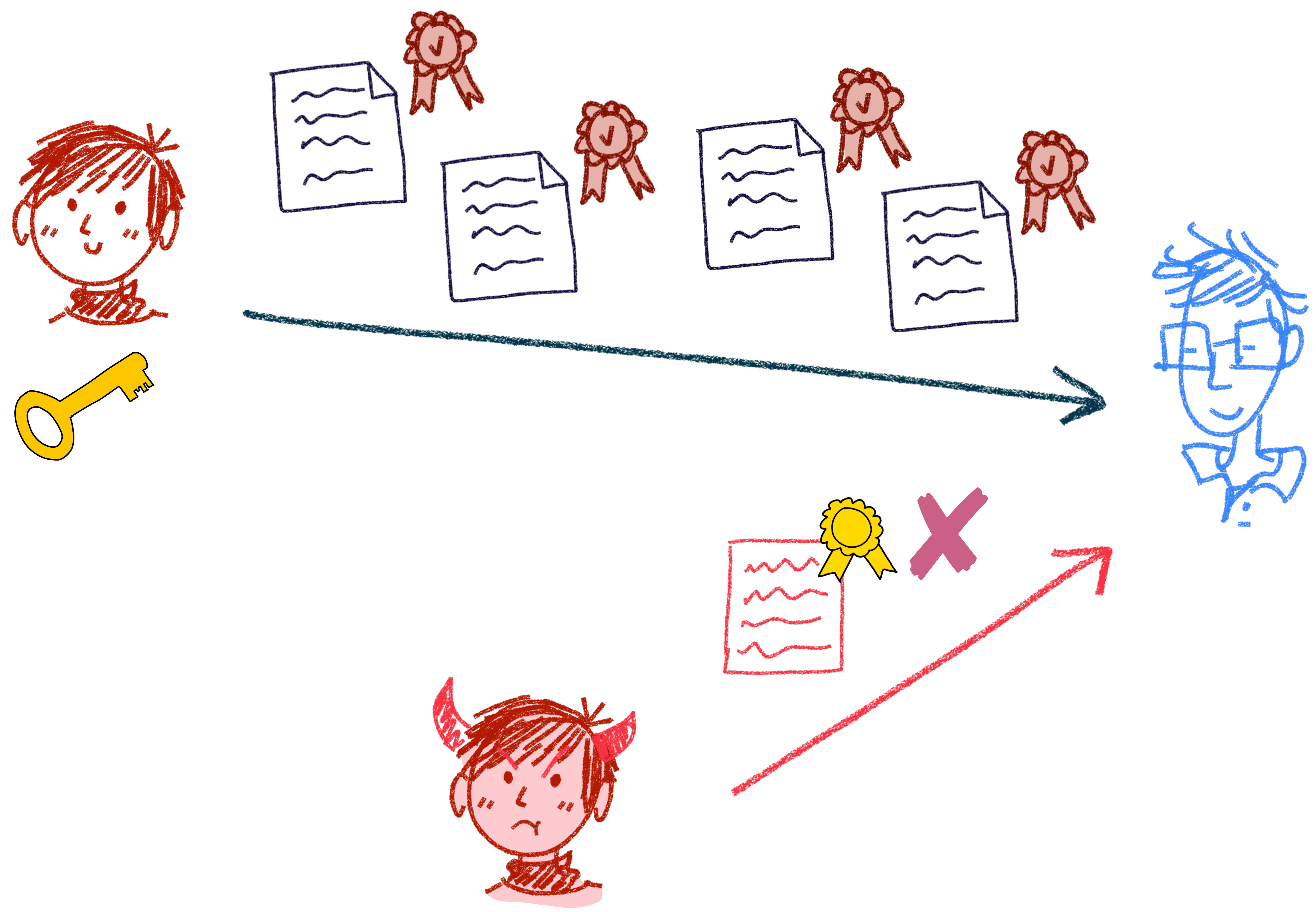
Signature Algorithms:

- KeyGen
- Sign
- Verify

Properties:

- Correctness
- Existential Unforgeability (EUF-CMA)

Digital Signatures



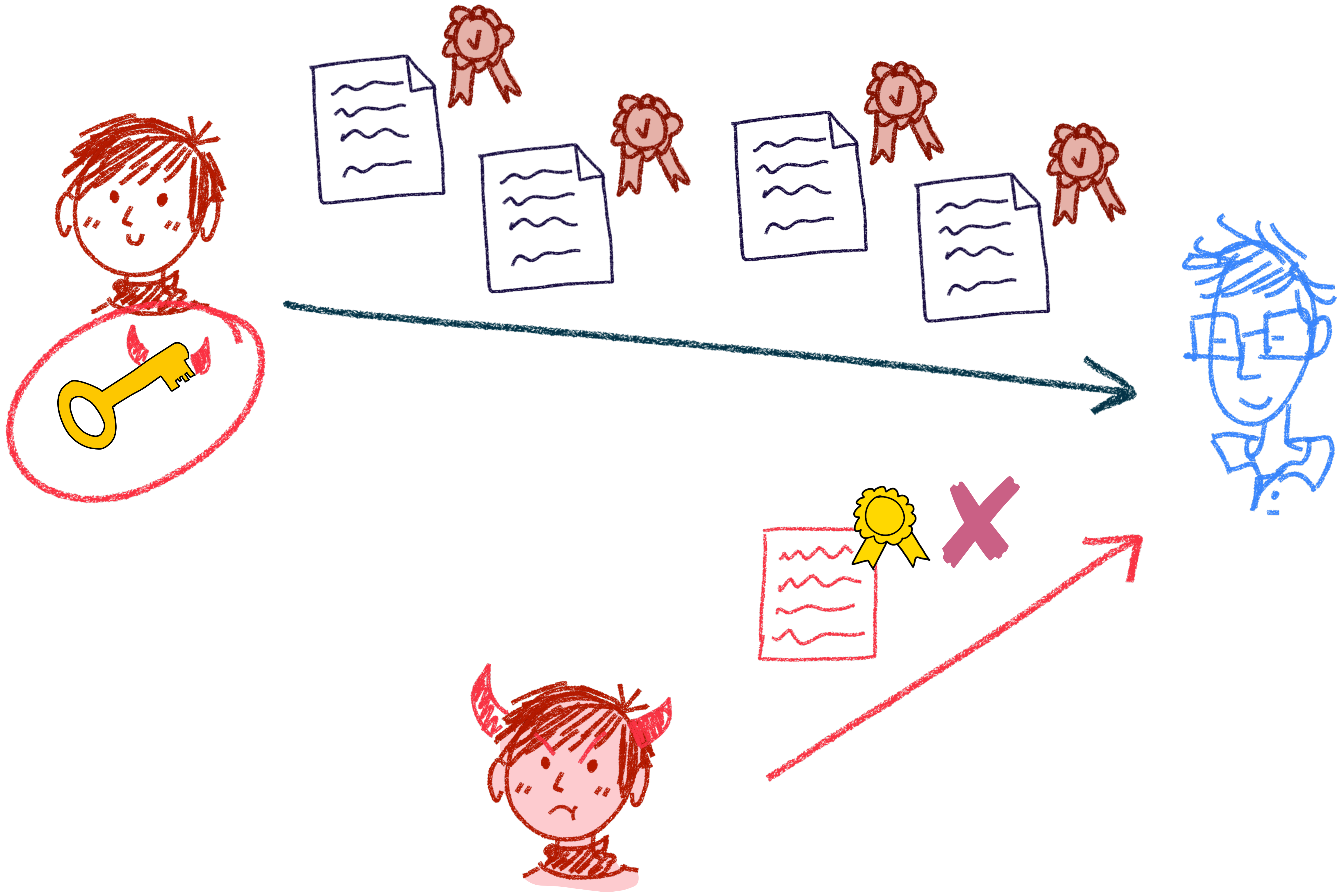
Signature Algorithms:

- KeyGen
- Sign
- Verify

Properties:

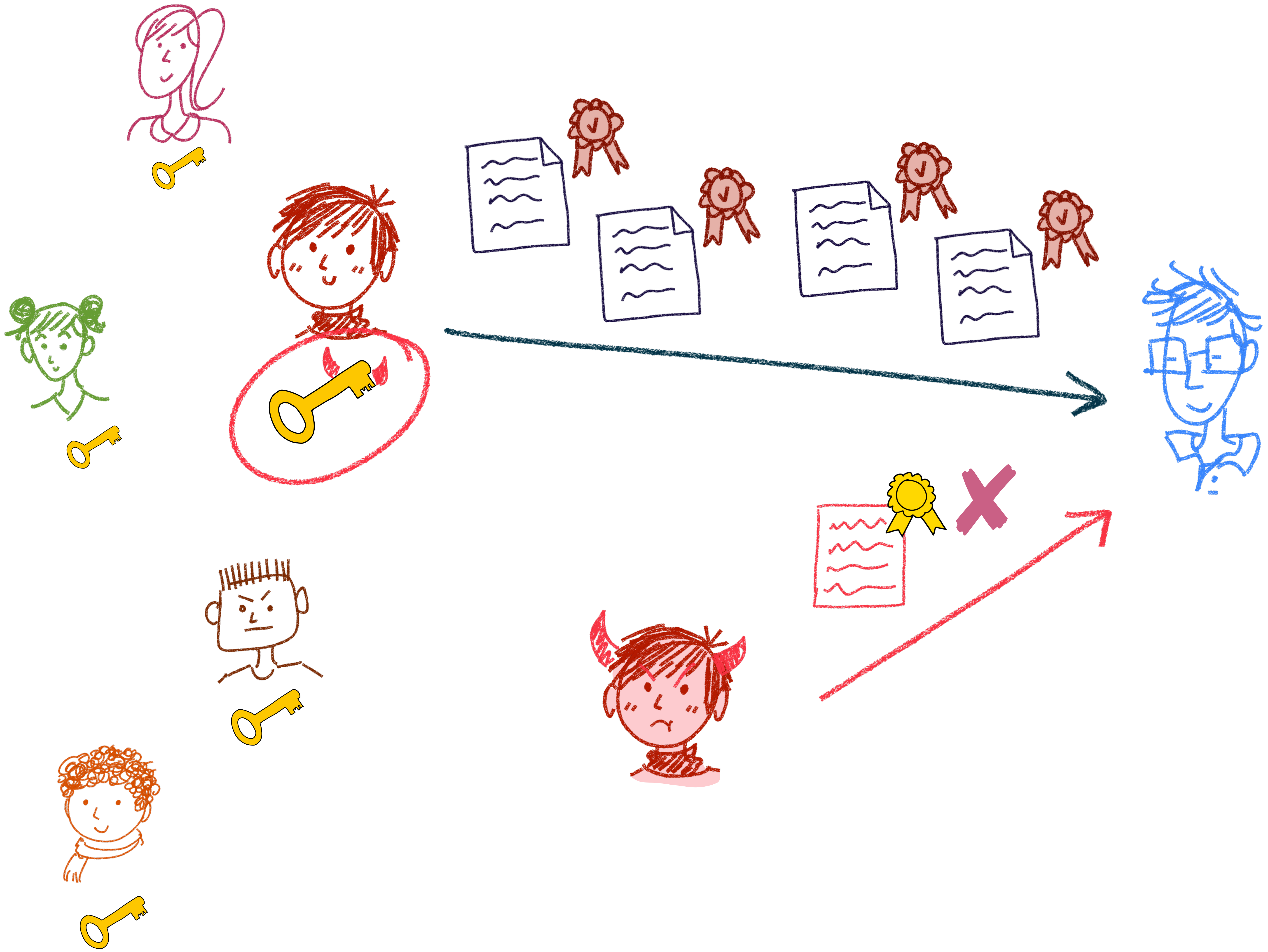
- Correctness
- Existential Unforgeability (EUF-CMA)

Properties not addressed by EUF-CMA



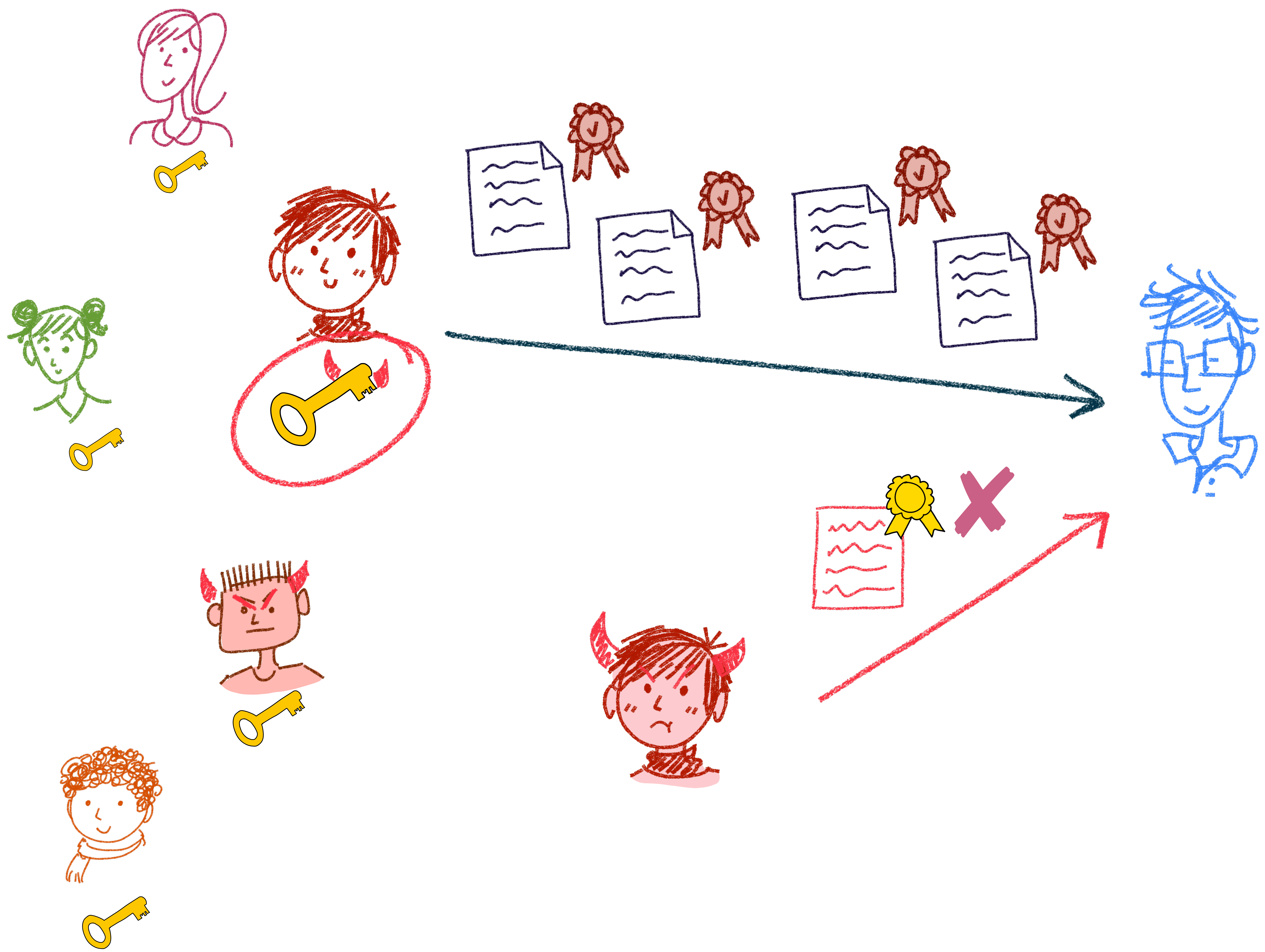
- Maliciously generated keys

Properties not addressed by EUF-CMA



- Maliciously generated keys
- Concurrency

Properties not addressed by EUF-CMA



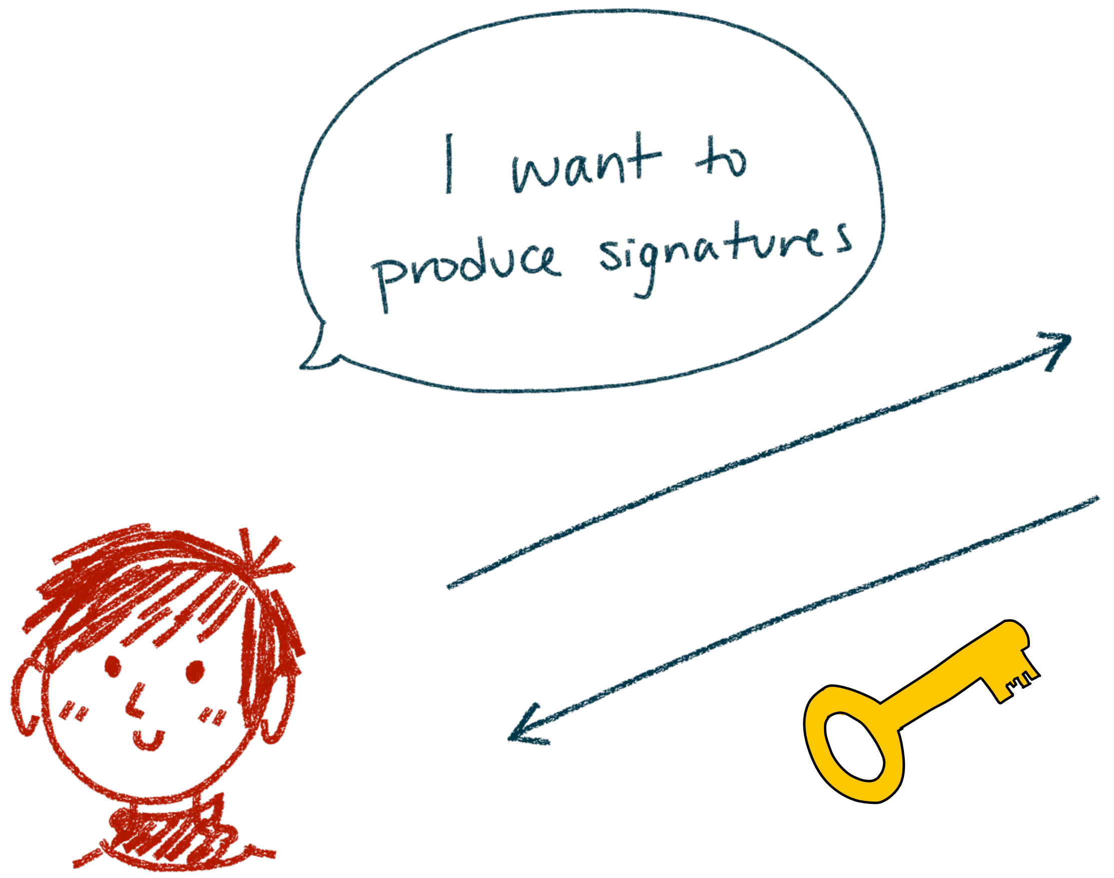
- Maliciously generated keys
- Concurrency
- Adaptive corruptions

Digital Signatures

Signature Functionality

Properties:
- Correctness
- Existential Unforgeability

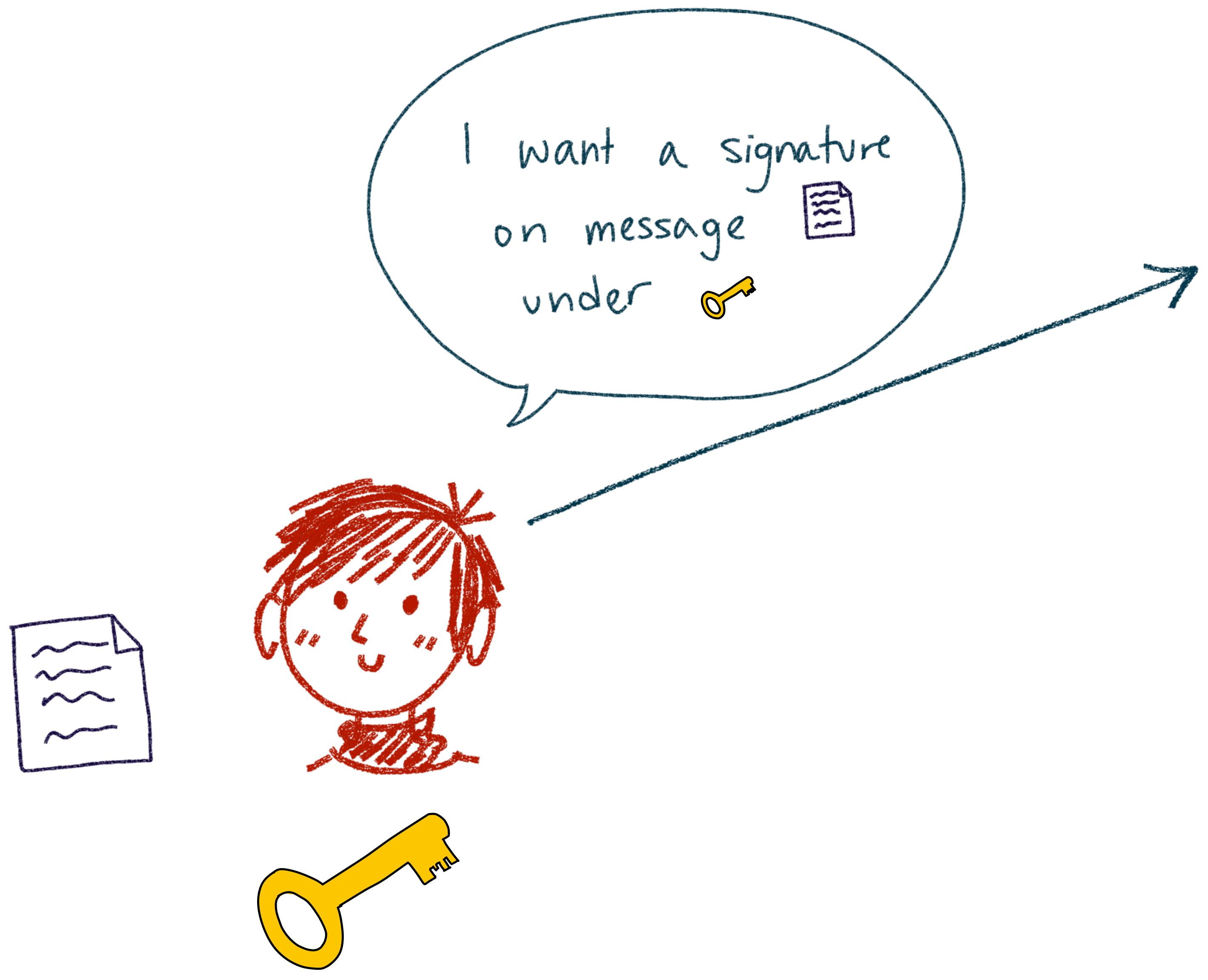
Digital Signatures



Signature Functionality

Properties:
- Correctness
- Existential Unforgeability

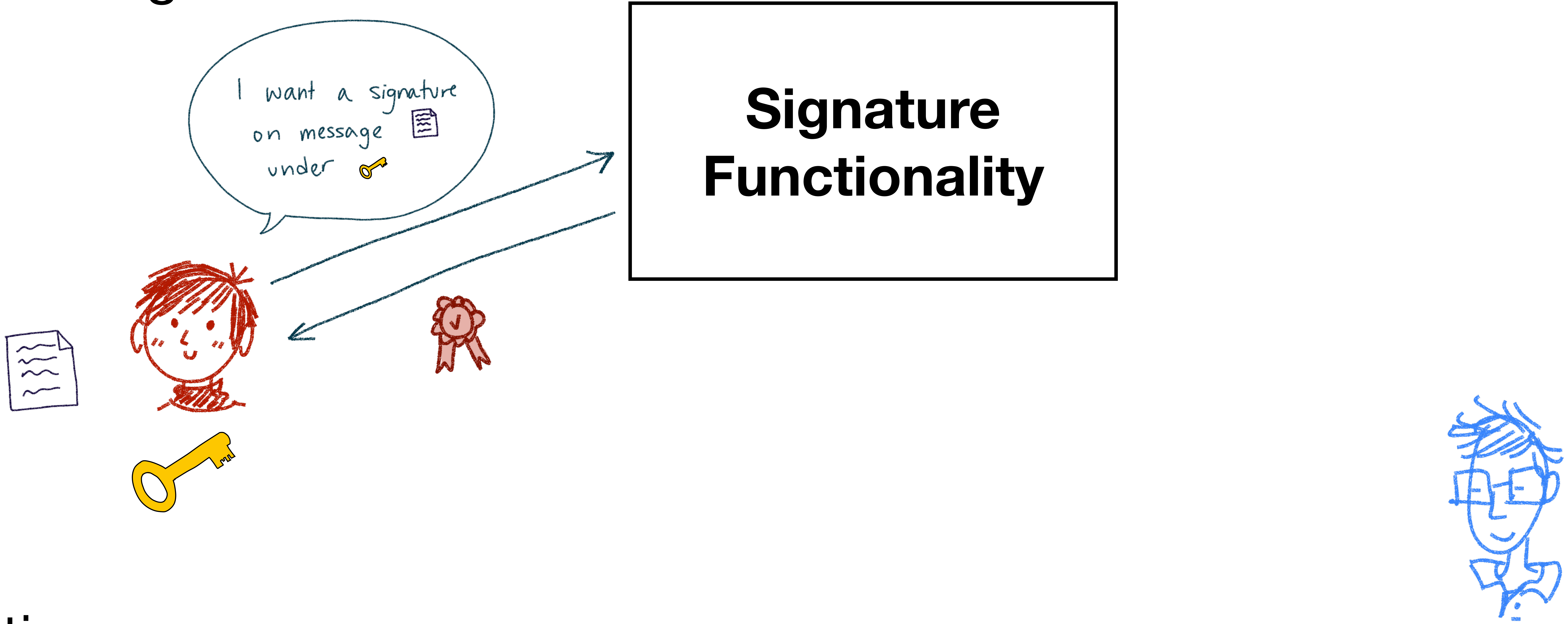
Digital Signatures



Signature Functionality

Properties:
- Correctness
- Existential Unforgeability

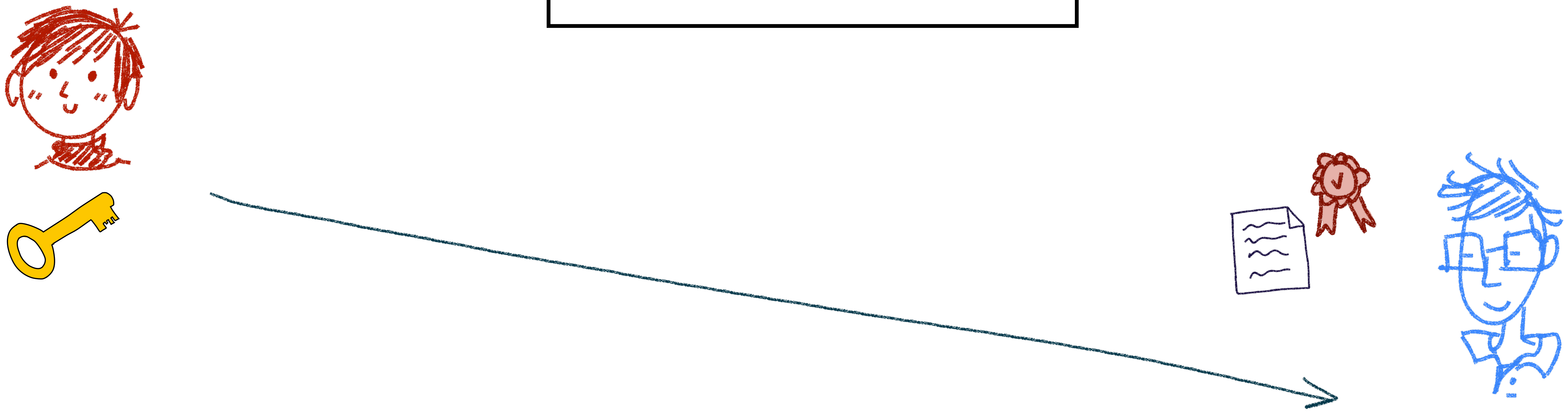
Digital Signatures



Properties:
- Correctness
- Existential Unforgeability

Digital Signatures

Signature Functionality



Properties:
- Correctness
- Existential Unforgeability

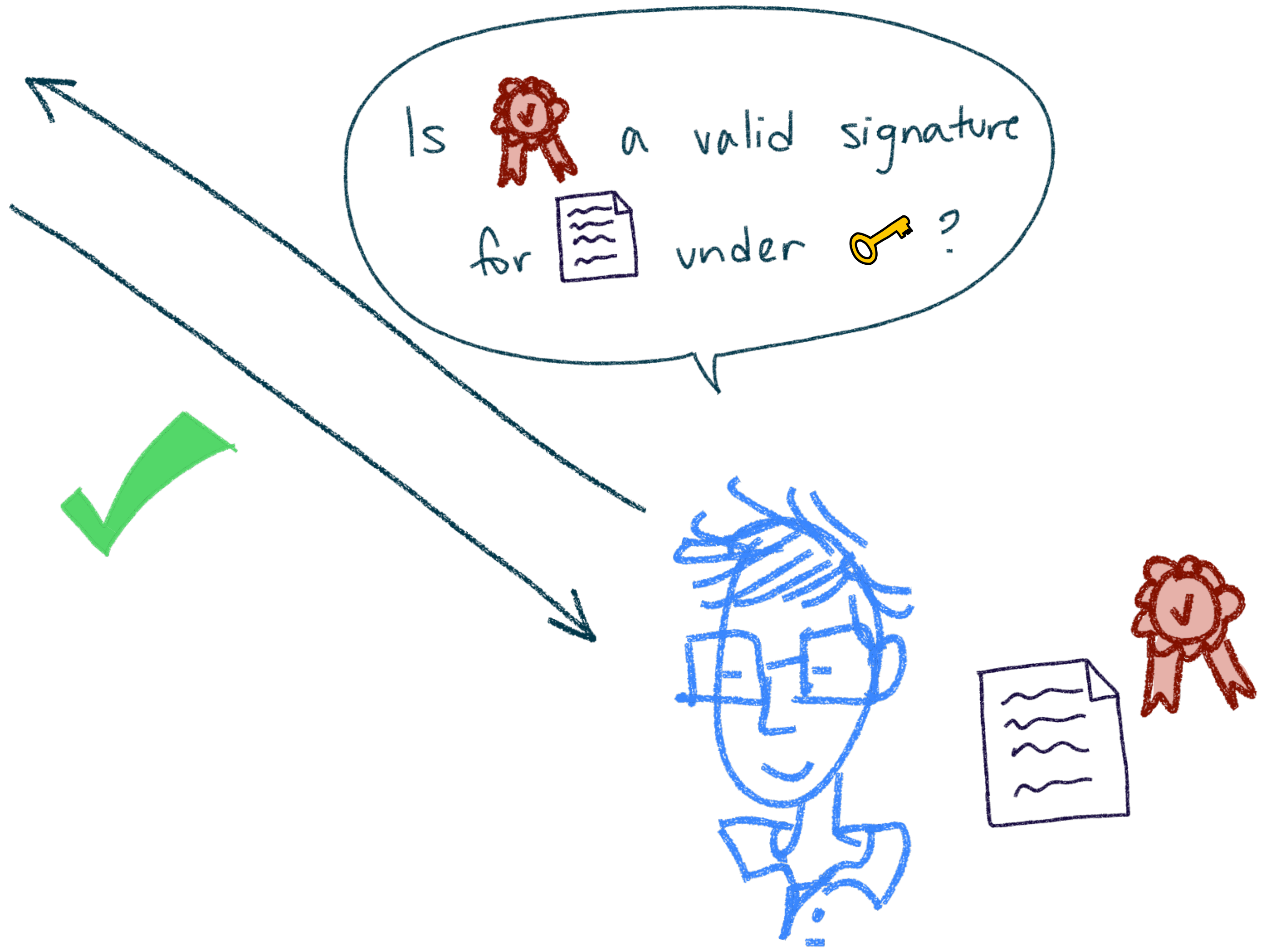
Digital Signatures

Signature Functionality

Properties:
- Correctness
- Existential Unforgeability

Digital Signatures

Signature Functionality

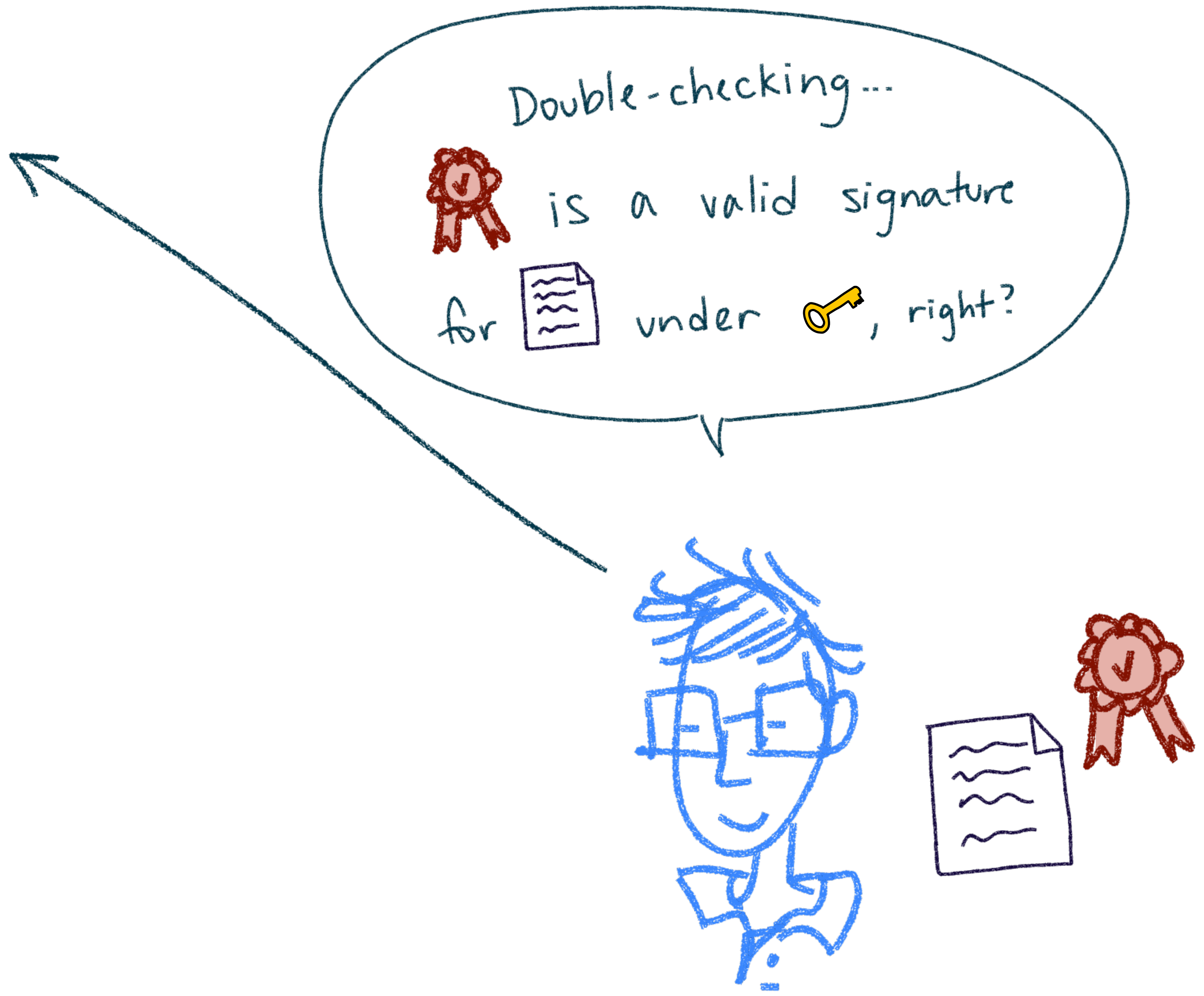


Properties:

- Correctness
- Existential Unforgeability

Digital Signatures

Signature Functionality



Properties:

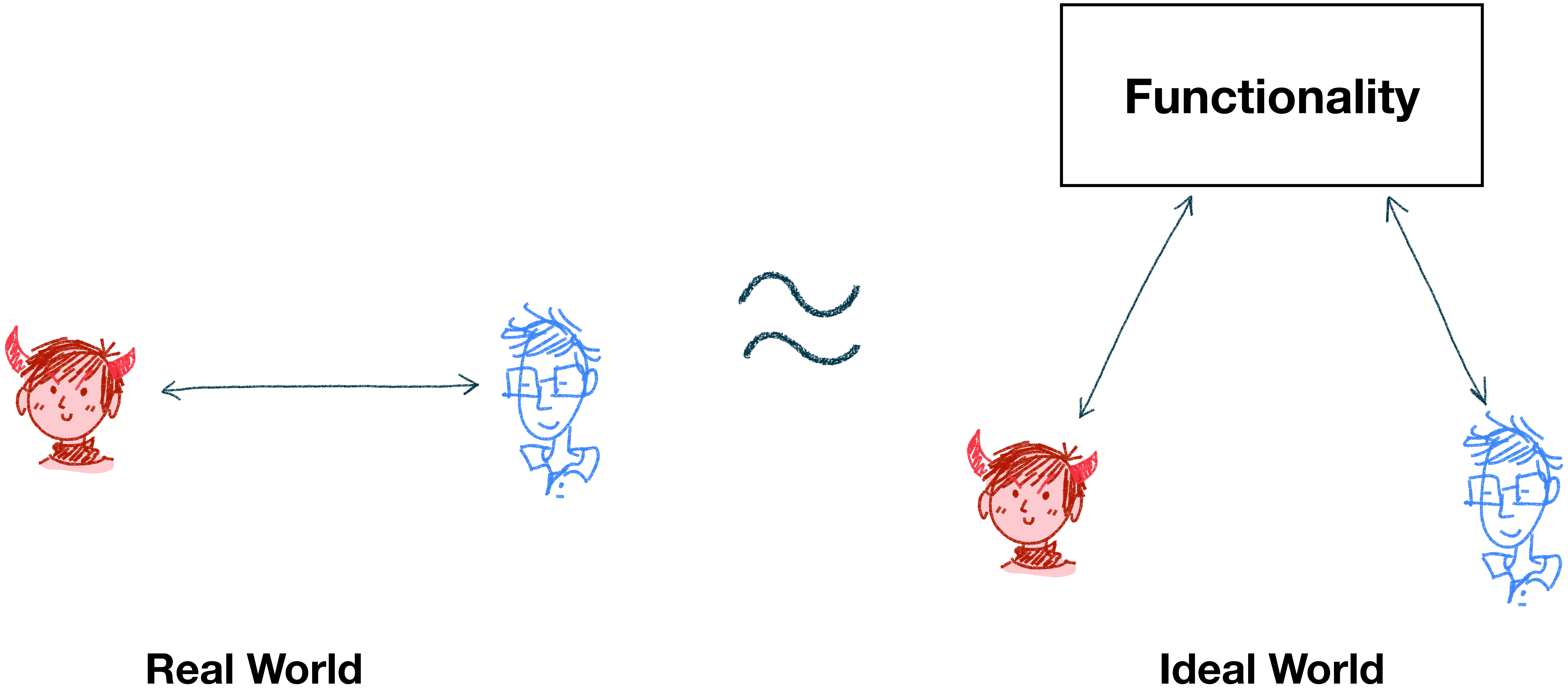
- Correctness
- Existential Unforgeability
- Consistency

Digital Signatures

Signature Functionality

Properties:
- Correctness
- Existential Unforgeability
- Consistency

Real/Ideal Paradigm



Universal Composition (UC)

- Framework for describing protocols and analyzing security
- Functionalities can be used as building blocks in larger protocols
- Composition theorem
  - If a scheme realizes a functionality, then that scheme can replace where the functionality is used as a sub-protocol
- Security maintained even with adversarially-controlled arbitrary concurrent sessions

Existing UC Signature Functionalities

- Many existing UC signature functionalities in the literature [C01,C04,CR03,BH04,GKZ10,CSV16,BCH+20,…]
    - Proven equivalent to EUF-CMA secure signatures
- However, issues arises when you plug it into a larger protocol
    - Adversary can force functionality into scenarios that don’t happen with secure signature schemes
    - Prior functionalities output an error and halt
        - Proofs don’t go through for protocols like Dolev-Strong broadcast!

Goal of this work (aka a sneak peek at the results)

- Can we define a signature functionality that works even when used in a larger protocol?
  - An *unstoppable* functionality that the adversary can't disable
- Can we show its usefulness by plugging it into Dolev-Strong broadcast and prove security?

Generations of Ideal Signatures

Pictured: Different generations of Pokemon that I don't expect anyone over the age of 30 to recognize

First generation: Pass control to the adversary

- Ask the adversary for responses to key generation and signing
  - Gives a lot of power to the adversary!
  - Adversary can choose to just never respond
  - Worse, adversary can force an error message

Functionality A.1. $\mathcal{F}_{\text{sig-1st}}$ (Example First-Generation Signature Functionality)

Key Generation. Upon receiving $(\texttt{keygen}, \mathsf{sid})$ from some party S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, hand $(\texttt{keygen}, \mathsf{sid})$ to the adversary. Upon receiving $(\texttt{VerificationKey}, \mathsf{sid}, v)$ from the adversary, output $(\texttt{VerificationKey}, \mathsf{sid}, v)$ to the caller S, and record the pair (S, v).

Sign. Upon receiving $(\texttt{sign}, \mathsf{sid}, m)$ from S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, send $(\texttt{sign}, \mathsf{sid}, m)$ to the adversary. Upon receiving $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to S, and record the entry $(m, \sigma, v, 1)$.

Verify. On receiving the value $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ from some party P, hand $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ to the adversary. Upon receiving $(\texttt{verified}, \mathsf{sid}, m, \phi)$ from the adversary, do:

1. If $v' = v$ and the entry $(m, \sigma, v, 1)$ is recorded, then set $b = 1$.
   (This condition guarantees completeness: If the verification key v' is the registered one and σ is a legitimately generated signature for m, then the verification succeeds)
2. Else, if $v' = v$, the signer is not corrupted, and no entry $(m, \sigma', v, 1)$ for any σ' is recorded, then set $b = 0$ and record the entry $(m, \sigma, v, 0)$.
   (This condition guarantees unforgeability: If v' is the registered one, the signer is not corrupted, and never signed m, then the verification fails.)
3. Else, if there is an entry (m, σ, v', b') recorded, then set $b = b'$.
   (This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)
4. Else, let $b = \phi$ and record the entry (m, σ, v', ϕ).

Return $(\texttt{verified}, \mathsf{sid}, m, b)$ to P.

First generation: Pass control to the adversary

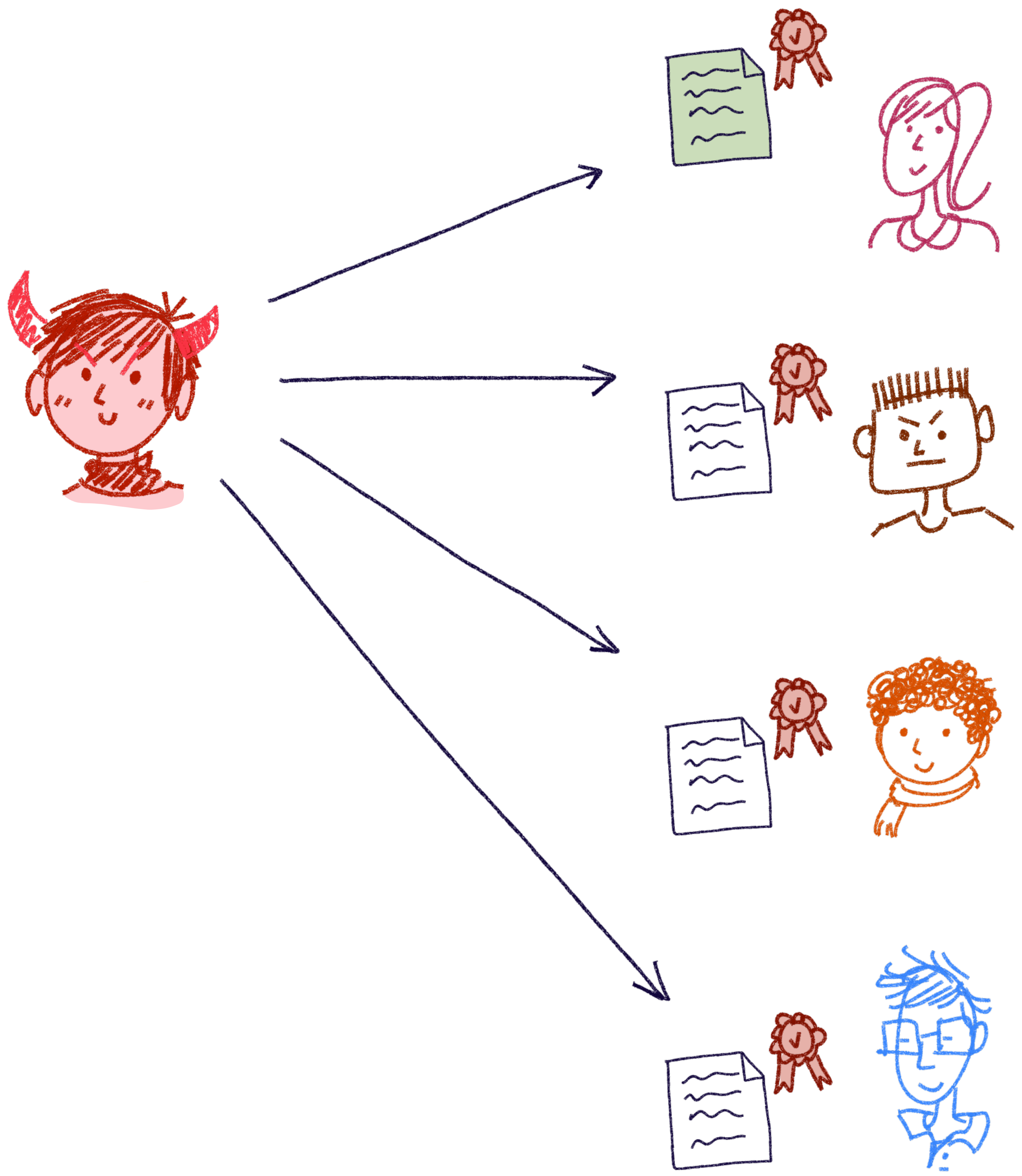
Functionality A.1. $\mathcal{F}_{\text{sig-1st}}$ (Example First-Generation Signature Functionality)

Sign. Upon receiving $(\texttt{sign}, \mathsf{sid}, m)$ from S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, send $(\texttt{sign}, \mathsf{sid}, m)$ to the adversary. Upon receiving $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to S, and record the entry $(m, \sigma, v, 1)$.

Verify. On receiving the value $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ from some party P, hand $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ to the adversary. Upon receiving $(\texttt{verified}, \mathsf{sid}, m, \phi)$ from the adversary, do:

1. If $v' = v$ and the entry $(m, \sigma, v, 1)$ is recorded, then set $b = 1$.
   (This condition guarantees completeness: If the verification key v' is the registered one and σ is a legitimately generated signature for m, then the verification succeeds)
2. Else, if $v' = v$, the signer is not corrupted, and no entry $(m, \sigma', v, 1)$ for any σ' is recorded, then set $b = 0$ and record the entry $(m, \sigma, v, 0)$.
   (This condition guarantees unforgeability: If v' is the registered one, the signer is not corrupted, and never signed m, then the verification fails.)

What can go wrong with first generation?



Functionality A.1. $\mathcal{F}_{\text{sig-1st}}$ (Example First-Generation Signature Functionality)

Key Generation. Upon receiving $(\texttt{keygen}, \mathsf{sid})$ from some party S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, hand $(\texttt{keygen}, \mathsf{sid})$ to the adversary. Upon receiving $(\texttt{VerificationKey}, \mathsf{sid}, v)$ from the adversary, output $(\texttt{VerificationKey}, \mathsf{sid}, v)$ to the caller S, and record the pair (S, v).

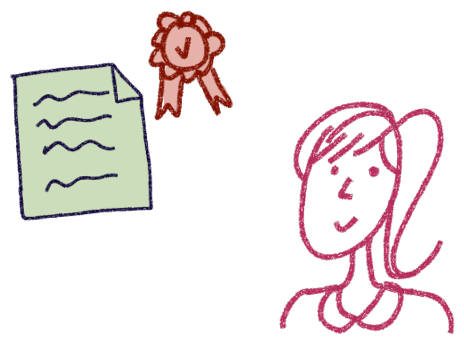
Sign. Upon receiving $(\texttt{sign}, \mathsf{sid}, m)$ from S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, send $(\texttt{sign}, \mathsf{sid}, m)$ to the adversary. Upon receiving $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to S, and record the entry $(m, \sigma, v, 1)$.

Verify. On receiving the value $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ from some party P, hand $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ to the adversary. Upon receiving $(\texttt{verified}, \mathsf{sid}, m, \phi)$ from the adversary, do:

1. If $v' = v$ and the entry $(m, \sigma, v, 1)$ is recorded, then set $b = 1$.
   (This condition guarantees completeness: If the verification key v' is the registered one and σ is a legitimately generated signature for m, then the verification succeeds)
2. Else, if $v' = v$, the signer is not corrupted, and no entry $(m, \sigma', v, 1)$ for any σ' is recorded, then set $b = 0$ and record the entry $(m, \sigma, v, 0)$.
   (This condition guarantees unforgeability: If v' is the registered one, the signer is not corrupted, and never signed m, then the verification fails.)
3. Else, if there is an entry (m, σ, v', b') recorded, then set $b = b'$.
   (This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)
4. Else, let $b = \phi$ and record the entry (m, σ, v', ϕ).

Return $(\texttt{verified}, \mathsf{sid}, m, b)$ to P.

What can go wrong with first generation?



Functionality A.1. $\mathcal{F}_{\text{sig-1st}}$ (Example First-Generation Signature Functionality)

Key Generation. Upon receiving $(\texttt{keygen}, \mathsf{sid})$ from some party S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, hand $(\texttt{keygen}, \mathsf{sid})$ to the adversary. Upon receiving $(\texttt{VerificationKey}, \mathsf{sid}, v)$ from the adversary, output $(\texttt{VerificationKey}, \mathsf{sid}, v)$ to the caller S, and record the pair (S, v).

Sign. Upon receiving $(\texttt{sign}, \mathsf{sid}, m)$ from S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, send $(\texttt{sign}, \mathsf{sid}, m)$ to the adversary. Upon receiving $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to S, and record the entry $(m, \sigma, v, 1)$.

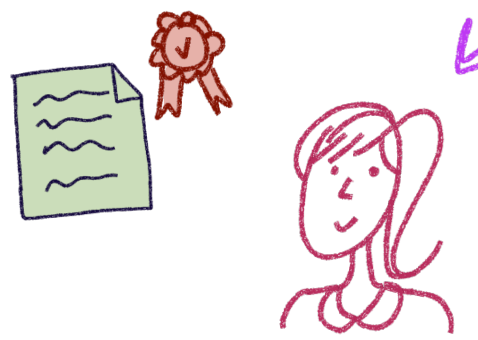
Verify. On receiving the value $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ from some party P, hand $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ to the adversary. Upon receiving $(\texttt{verified}, \mathsf{sid}, m, \phi)$ from the adversary, do:

1. If $v' = v$ and the entry $(m, \sigma, v, 1)$ is recorded, then set $b = 1$.
   (This condition guarantees completeness: If the verification key v' is the registered one and σ is a legitimately generated signature for m, then the verification succeeds)
2. Else, if $v' = v$, the signer is not corrupted, and no entry $(m, \sigma', v, 1)$ for any σ' is recorded, then set $b = 0$ and record the entry $(m, \sigma, v, 0)$.
   (This condition guarantees unforgeability: If v' is the registered one, the signer is not corrupted, and never signed m, then the verification fails.)
3. Else, if there is an entry (m, σ, v', b') recorded, then set $b = b'$.
   (This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)
4. Else, let $b = \phi$ and record the entry (m, σ, v', ϕ).

Return $(\texttt{verified}, \mathsf{sid}, m, b)$ to P.

What can go wrong with first generation?

Alice wants a signature on what she received



Functionality A.1. $\mathcal{F}_{\text{sig-1st}}$ (Example First-Generation Signature Functionality)

Key Generation. Upon receiving $(\texttt{keygen}, \mathsf{sid})$ from some party S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, hand $(\texttt{keygen}, \mathsf{sid})$ to the adversary. Upon receiving $(\texttt{VerificationKey}, \mathsf{sid}, v)$ from the adversary, output $(\texttt{VerificationKey}, \mathsf{sid}, v)$ to the caller S, and record the pair (S, v).

Sign. Upon receiving $(\texttt{sign}, \mathsf{sid}, m)$ from S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, send $(\texttt{sign}, \mathsf{sid}, m)$ to the adversary. Upon receiving $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to S, and record the entry $(m, \sigma, v, 1)$.

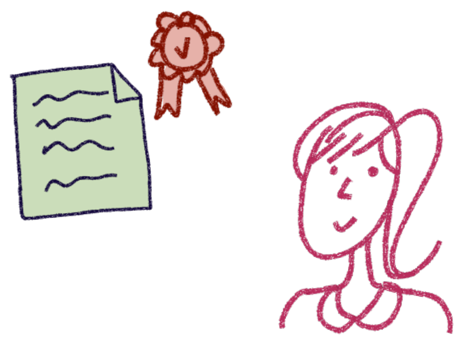
Verify. On receiving the value $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ from some party P, hand $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ to the adversary. Upon receiving $(\texttt{verified}, \mathsf{sid}, m, \phi)$ from the adversary, do:

1. If $v' = v$ and the entry $(m, \sigma, v, 1)$ is recorded, then set $b = 1$.
   (This condition guarantees completeness: If the verification key v' is the registered one and σ is a legitimately generated signature for m, then the verification succeeds)
2. Else, if $v' = v$, the signer is not corrupted, and no entry $(m, \sigma', v, 1)$ for any σ' is recorded, then set $b = 0$ and record the entry $(m, \sigma, v, 0)$.
   (This condition guarantees unforgeability: If v' is the registered one, the signer is not corrupted, and never signed m, then the verification fails.)
3. Else, if there is an entry (m, σ, v', b') recorded, then set $b = b'$.
   (This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)
4. Else, let $b = \phi$ and record the entry (m, σ, v', ϕ).

Return $(\texttt{verified}, \mathsf{sid}, m, b)$ to P.

What can go wrong with first generation?

Alice wants a signature on what she received



$(\texttt{sign}, \mathsf{sid}, m)$

Functionality A.1. $\mathcal{F}_{\text{sig-1st}}$ (Example First-Generation Signature Functionality)

Sign. Upon receiving $(\texttt{sign}, \mathsf{sid}, m)$ from S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, send $(\texttt{sign}, \mathsf{sid}, m)$ to the adversary. Upon receiving $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to S, and record the entry $(m, \sigma, v, 1)$.

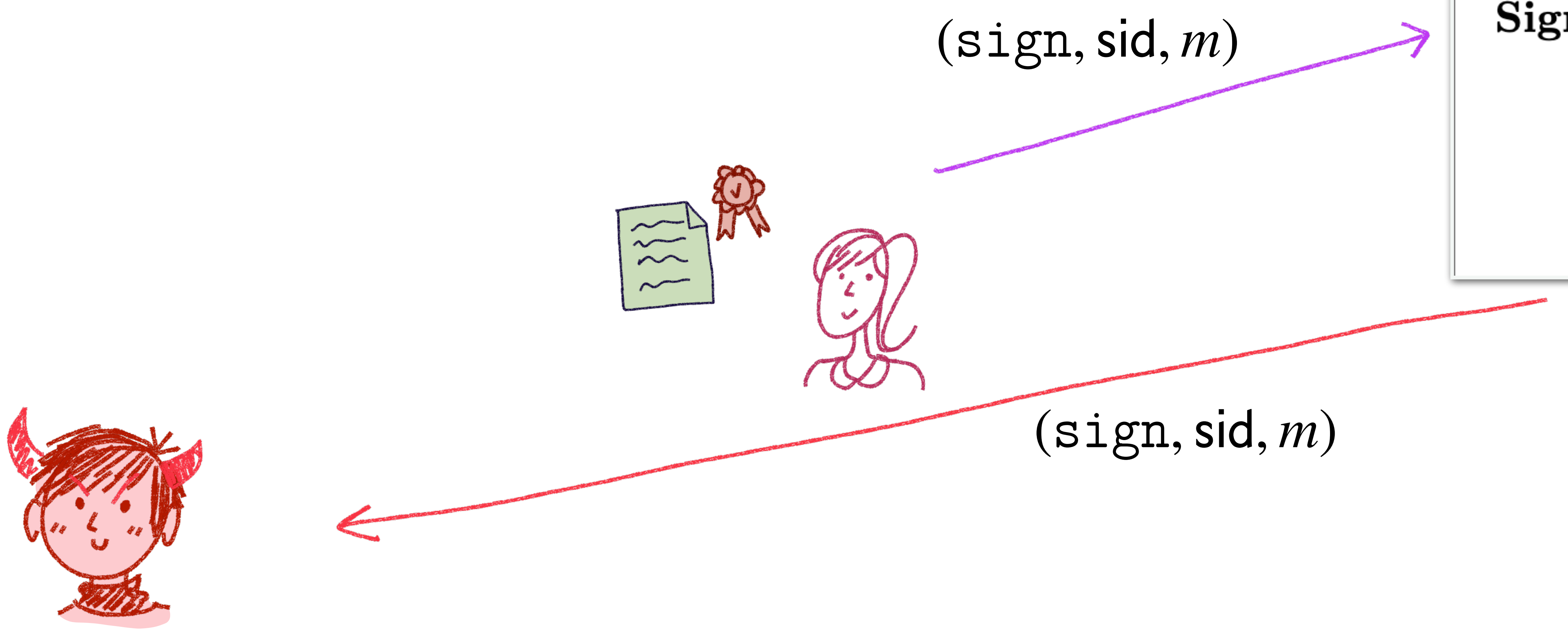
then ignore this request. Else, send $(\texttt{sign}, \mathsf{sid}, m)$ to the adversary. Upon receiving $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to S, and record the entry $(m, \sigma, v, 1)$.

Verify. On receiving the value $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ from some party P, hand $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ to the adversary. Upon receiving $(\texttt{verified}, \mathsf{sid}, m, \phi)$ from the adversary, do:

1. If $v' = v$ and the entry $(m, \sigma, v, 1)$ is recorded, then set $b = 1$.
   (This condition guarantees completeness: If the verification key v' is the registered one and σ is a legitimately generated signature for m, then the verification succeeds)
2. Else, if $v' = v$, the signer is not corrupted, and no entry $(m, \sigma', v, 1)$ for any σ' is recorded, then set $b = 0$ and record the entry $(m, \sigma, v, 0)$.
   (This condition guarantees unforgeability: If v' is the registered one, the signer is not corrupted, and never signed m, then the verification fails.)
3. Else, if there is an entry (m, σ, v', b') recorded, then set $b = b'$.
   (This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)
4. Else, let $b = \phi$ and record the entry (m, σ, v', ϕ).

Return $(\texttt{verified}, \mathsf{sid}, m, b)$ to P.

What can go wrong with first generation?



Functionality asks Adv for the signature

Sign. Upon receiving $(\texttt{sign}, \mathsf{sid}, m)$ from S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, send $(\texttt{sign}, \mathsf{sid}, m)$ to the adversary. Upon receiving $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to S, and record the entry $(m, \sigma, v, 1)$.

Functionality A.1. ...ionality)

then ignore this request. Else, send $(\texttt{sign}, \mathsf{sid}, m)$ to the adversary. Upon receiving $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to S, and record the entry $(m, \sigma, v, 1)$.

Verify. On receiving the value $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ from some party P, hand $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ to the adversary. Upon receiving $(\texttt{verified}, \mathsf{sid}, m, \phi)$ from the adversary, do:

1. If $v' = v$ and the entry $(m, \sigma, v, 1)$ is recorded, then set $b = 1$.
   (This condition guarantees completeness: If the verification key v' is the registered one and σ is a legitimately generated signature for m, then the verification succeeds)
2. Else, if $v' = v$, the signer is not corrupted, and no entry $(m, \sigma', v, 1)$ for any σ' is recorded, then set $b = 0$ and record the entry $(m, \sigma, v, 0)$.
   (This condition guarantees unforgeability: If v' is the registered one, the signer is not corrupted, and never signed m, then the verification fails.)
3. Else, if there is an entry (m, σ, v', b') recorded, then set $b = b'$.
   (This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)
4. Else, let $b = \phi$ and record the entry (m, σ, v', ϕ).

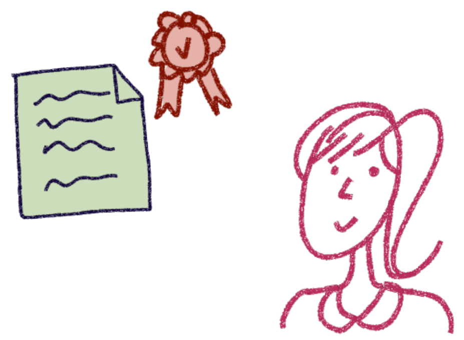
Return $(\texttt{verified}, \mathsf{sid}, m, b)$ to P.

What can go wrong with first generation?

Functionality asks Adv for the signature

Functionality A.1. ...ionality)

$(\texttt{sign}, \mathsf{sid}, m)$



$(\texttt{sign}, \mathsf{sid}, m)$

$(\texttt{verify}, \mathsf{sid}, m, \sigma, v)$

Sign. Upon receiving $(\texttt{sign}, \mathsf{sid}, m)$ from S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, send $(\texttt{sign}, \mathsf{sid}, m)$ to the adversary. Upon receiving $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to S, and record the entry $(m, \sigma, v, 1)$.

then ignore this request. Else, send $(\texttt{sign}, \mathsf{sid}, m)$ to the adversary. Upon receiving $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to S, and record the entry $(m, \sigma, v, 1)$.

Adv uses verify interface to get (m, σ) registered as a bad signature pair

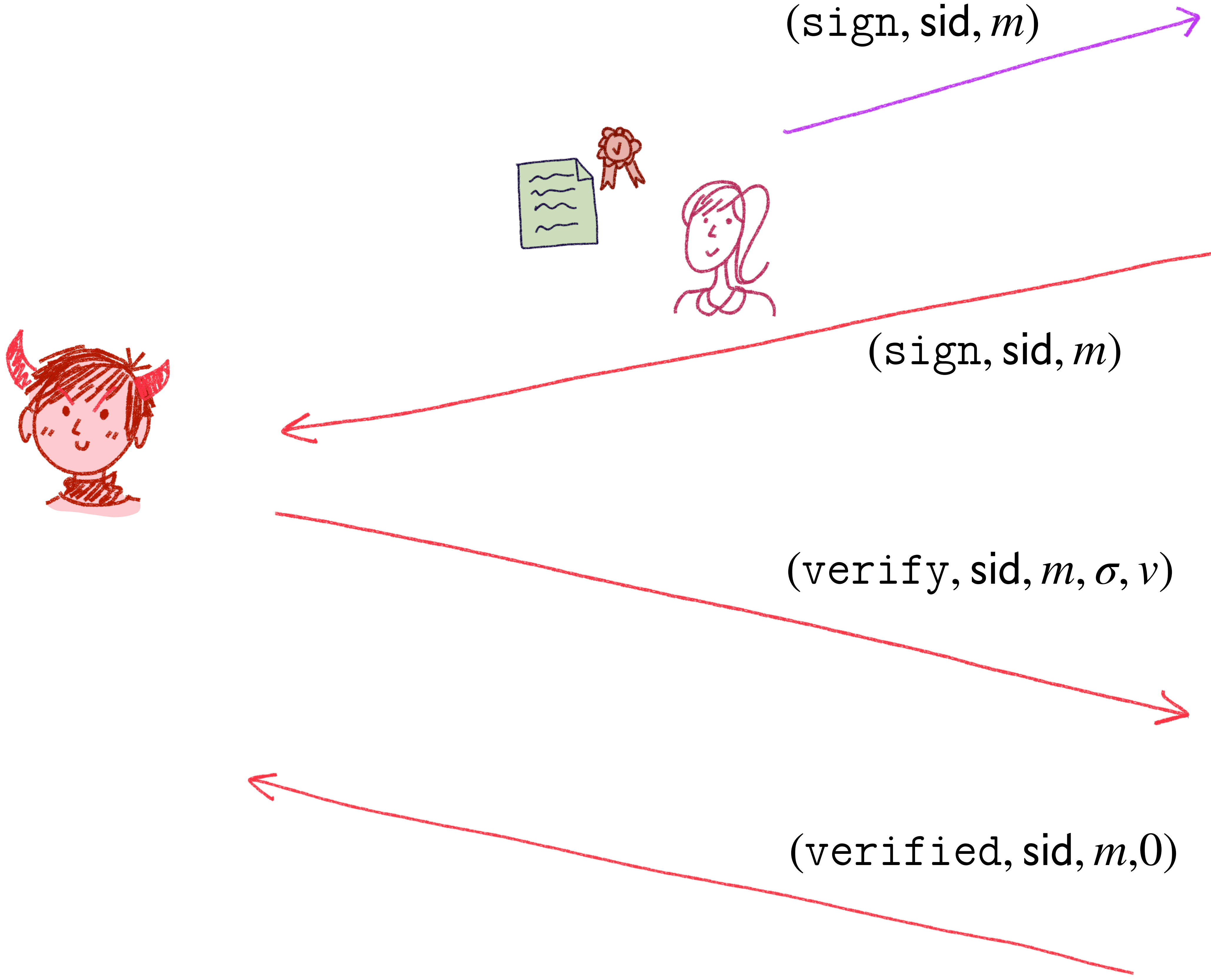
Verify. On receiving the value $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ from some party P, hand $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ to the adversary. Upon receiving $(\texttt{verified}, \mathsf{sid}, m, \phi)$ from the adversary, do:

1. If $v' = v$ and the entry $(m, \sigma, v, 1)$ is recorded, then set $b = 1$.
   (This condition guarantees completeness: If the verification key v' is the registered one and σ is a legitimately generated signature for m, then the verification succeeds)
2. Else, if $v' = v$, the signer is not corrupted, and no entry $(m, \sigma', v, 1)$ for any σ' is recorded, then set $b = 0$ and record the entry $(m, \sigma, v, 0)$.
   (This condition guarantees unforgeability: If v' is the registered one, the signer is not corrupted, and never signed m, then the verification fails.)

What can go wrong with first generation?

Functionality asks Adv for the signature

Functionality A.1. … ionality)



Sign. Upon receiving $(\texttt{sign}, \mathsf{sid}, m)$ from S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, send $(\texttt{sign}, \mathsf{sid}, m)$ to the adversary. Upon receiving $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to S, and record the entry $(m, \sigma, v, 1)$.

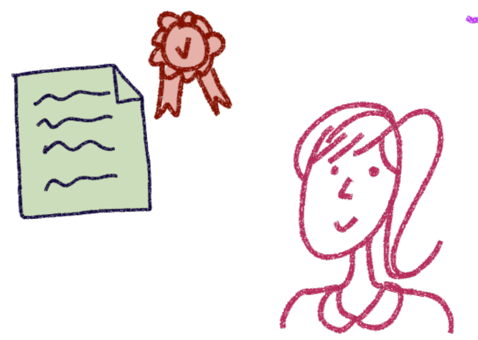
Adv uses verify interface to get (m,σ) registered as a bad signature pair

Verify. On receiving the value $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ from some party P, hand $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ to the adversary. Upon receiving $(\texttt{verified}, \mathsf{sid}, m, \phi)$ from the adversary, do:

1. If $v' = v$ and the entry $(m, \sigma, v, 1)$ is recorded, then set $b = 1$.
   (This condition guarantees completeness: If the verification key v' is the registered one and σ is a legitimately generated signature for m, then the verification succeeds)
2. Else, if $v' = v$, the signer is not corrupted, and no entry $(m, \sigma', v, 1)$ for any σ' is recorded, then set $b = 0$ and record the entry $(m, \sigma, v, 0)$.
   (This condition guarantees unforgeability: If v' is the registered one, the signer is not corrupted, and never signed m, then the verification fails.)

What can go wrong with first generation?

$(\texttt{sign}, \mathsf{sid}, m)$



$(\texttt{sign}, \mathsf{sid}, m)$

$(\texttt{signature}, \mathsf{sid}, m, \sigma)$

Functionality asks Adv for the signature

Functionality A.1. ionality)

Sign. Upon receiving $(\texttt{sign}, \mathsf{sid}, m)$ from S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, send $(\texttt{sign}, \mathsf{sid}, m)$ to the adversary. Upon receiving $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to S, and record the entry $(m, \sigma, v, 1)$.

then ignore this request. Else, send $(\texttt{sign}, \mathsf{sid}, m)$ to the adversary. Upon receiving $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to S, and record the entry $(m, \sigma, v, 1)$.

Verify. hand (v m the ad

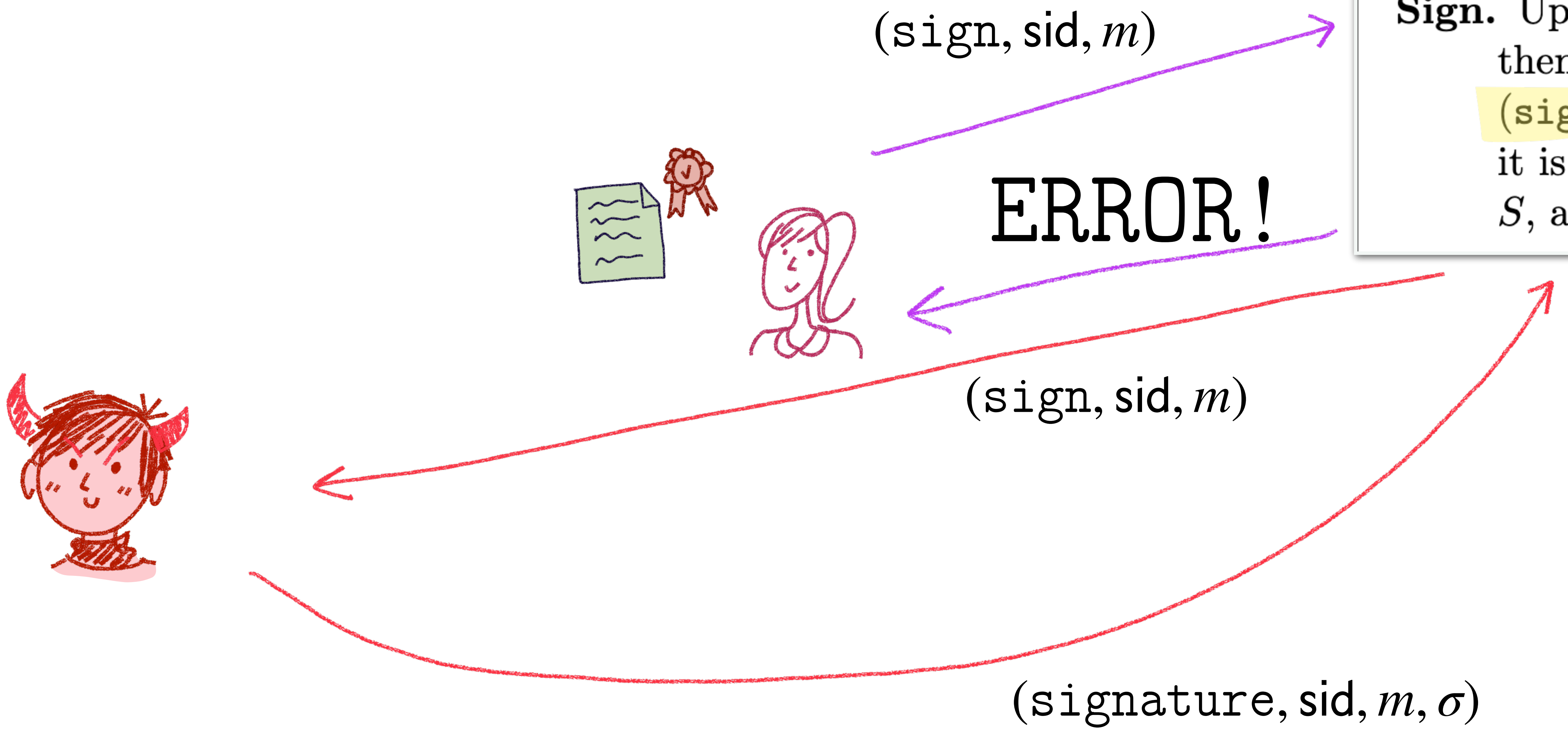
Adv uses verify interface to get (m, σ) registered as a bad signature pair

(This condition guarantees completeness: If the verification key v' is the registered one and σ is a legitimately generated signature for m, then the verification succeeds)

2. Else, if $v' = v$, the signer is not corrupted, and no entry $(m, \sigma', v, 1)$ for any σ' is recorded, then set $b = 0$ and record the entry $(m, \sigma, v, 0)$.
   (This condition guarantees unforgeability: If v' is the registered one, the signer is not corrupted, and never signed m, then the verification fails.)
3. Else, if there is an entry (m, σ, v', b') recorded, then set $b = b'$.
   (This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)
4. Else, let $b = \phi$ and record the entry (m, σ, v', ϕ).

Return $(\texttt{verified}, \mathsf{sid}, m, b)$ to P.

What can go wrong with first generation?



Functionality asks Adv for the signature

Functionality A.1. ionality)

Sign. Upon receiving $(\texttt{sign}, \mathsf{sid}, m)$ from S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, send $(\texttt{sign}, \mathsf{sid}, m)$ to the adversary. Upon receiving $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to S, and record the entry $(m, \sigma, v, 1)$.

then ignore this request. Else, send $(\texttt{sign}, \mathsf{sid}, m)$ to the adversary. Upon receiving $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to S, and record the entry $(m, \sigma, v, 1)$.

Verify. hand (v m the ad

Adv uses verify interface to get (m, σ) registered as a bad signature pair

(This condition guarantees completeness: If the verification key v' is the registered one and σ is a legitimately generated signature for m, then the verification succeeds)

2. Else, if $v' = v$, the signer is not corrupted, and no entry $(m, \sigma', v, 1)$ for any σ' is recorded, then set $b = 0$ and record the entry $(m, \sigma, v, 0)$.
(This condition guarantees unforgeability: If v' is the registered one, the signer is not corrupted, and never signed m, then the verification fails.)
3. Else, if there is an entry (m, σ, v', b') recorded, then set $b = b'$.
(This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)
4. Else, let $b = \phi$ and record the entry (m, σ, v', ϕ).

Return $(\texttt{verified}, \mathsf{sid}, m, b)$ to P.

Second generation: Adversary supplies algorithms

- Use algorithms given by the Adversary
  - Works when supplied with honest algorithms
  - What if the adversary supplies bad algorithms?

Functionality A.2. $\mathcal{F}_{\text{sig-2nd}}$ (Example Second-Generation Signature Functionality)

Key Generation. Upon receiving $(\texttt{keygen}, \mathsf{sid})$ from some party S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, hand $(\texttt{keygen}, \mathsf{sid})$ to the adversary. Upon receiving $(\texttt{algs}, \mathsf{sid}, s, v)$ from the adversary, where s is a description of a PPT ITM, and v is a description of a *deterministic* polytime ITM, output $(\texttt{VerificationAlgorithm}, \mathsf{sid}, v)$ to S.

Sign. Upon receiving $(\texttt{sign}, \mathsf{sid}, m)$ from S, let $\sigma = s(m)$, and verify that $v(m, \sigma) = 1$. If so, then output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to the caller P_i and record the entry (m, σ). Else, output an error message to S and halt.

Verify. On receiving the value $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ from some party V do: If $v' = v$, the signer is not corrupted, $v(m, \sigma) = 1$, and no entry (m, σ') for any σ' is recorded, then output an error message to S and halt. Else, output $(\texttt{verified}, \mathsf{sid}, m, v'(m, \sigma))$ to V.

What can go wrong with second generation?

Functionality A.2. $\mathcal{F}_{\text{sig-2nd}}$ (Example Second-Generation Signature Functionality)

Key Generation. Upon receiving $(\mathtt{keygen}, \mathsf{sid})$ from some party S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, hand $(\mathtt{keygen}, \mathsf{sid})$ to the adversary. Upon receiving $(\mathtt{algs}, \mathsf{sid}, s, v)$ from the adversary, where s is a description of a PPT ITM, and v is a description of a *deterministic* polytime ITM, output $(\mathtt{VerificationAlgorithm}, \mathsf{sid}, v)$ to S.

Sign. Upon receiving $(\mathtt{sign}, \mathsf{sid}, m)$ from S, let $\sigma = s(m)$, and verify that $v(m, \sigma) = 1$. If so, then output $(\mathtt{signature}, \mathsf{sid}, m, \sigma)$ to the caller P_i and record the entry (m, σ). Else, output an error message to S and halt.

Verify. On receiving the value $(\mathtt{verify}, \mathsf{sid}, m, \sigma, v')$ from some party V do: If $v' = v$, the signer is not corrupted, $v(m, \sigma) = 1$, and no entry (m, σ') for any σ' is recorded, then output an error message to S and halt. Else, output $(\mathtt{verified}, \mathsf{sid}, m, v'(m, \sigma))$ to V.

What can go wrong with second generation?

Functionality asks Adv for signing and verification algs

Functionality A.2. $\mathcal{F}_{\text{sig-2nd}}$ (Example

Adv can hard-code verify to always fail on strings of a specific form (e.g., fail if message contains Alice's ID)

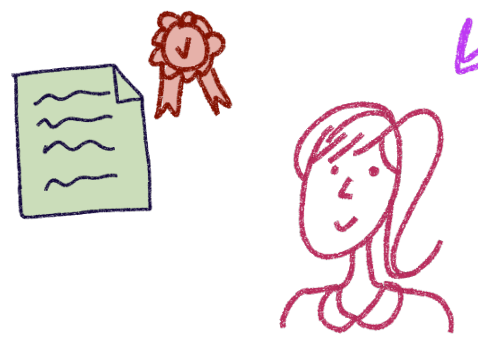
Key Generation. Upon receiving $(\texttt{keygen}, \mathsf{sid})$ from some party S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, hand $(\texttt{keygen}, \mathsf{sid})$ to the adversary. Upon receiving $(\texttt{algs}, \mathsf{sid}, s, v)$ from the adversary, where s is a description of a PPT ITM, and v is a description of a *deterministic* polytime ITM, output $(\texttt{VerificationAlgorithm}, \mathsf{sid}, v)$ to S.

Sign. Upon receiving $(\texttt{sign}, \mathsf{sid}, m)$ from S, let $\sigma = s(m)$, and verify that $v(m, \sigma) = 1$. If so, then output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to the caller P_i and record the entry (m, σ). Else, output an error message to S and halt.

Verify. On receiving the value $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ from some party V do: If $v' = v$, the signer is not corrupted, $v(m, \sigma) = 1$, and no entry (m, σ') for any σ' is recorded, then output an error message to S and halt. Else, output $(\texttt{verified}, \mathsf{sid}, m, v'(m, \sigma))$ to V.

What can go wrong with second generation?

Alice wants a signature on what she received



Functionality asks Adv for signing and verification algs

Adv can hard-code verify to always fail on strings of a specific form (e.g., fail if message contains Alice's ID)

Functionality A.2. $\mathcal{F}_{\text{sig-2nd}}$ (Example

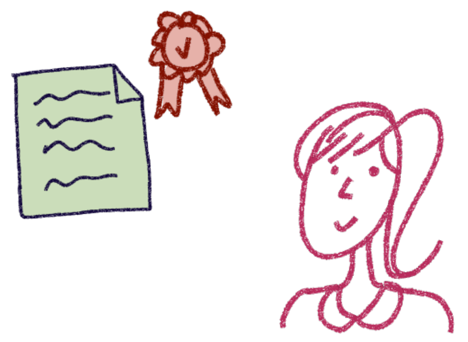
Key Generation. Upon receiving $(\texttt{keygen}, \mathsf{sid})$ from some party S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, hand $(\texttt{keygen}, \mathsf{sid})$ to the adversary. Upon receiving $(\texttt{algs}, \mathsf{sid}, s, v)$ from the adversary, where s is a description of a PPT ITM, and v is a description of a *deterministic* polytime ITM, output $(\texttt{VerificationAlgorithm}, \mathsf{sid}, v)$ to S.

Sign. Upon receiving $(\texttt{sign}, \mathsf{sid}, m)$ from S, let $\sigma = s(m)$, and verify that $v(m, \sigma) = 1$. If so, then output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to the caller P_i and record the entry (m, σ). Else, output an error message to S and halt.

Verify. On receiving the value $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ from some party V do: If $v' = v$, the signer is not corrupted, $v(m, \sigma) = 1$, and no entry (m, σ') for any σ' is recorded, then output an error message to S and halt. Else, output $(\texttt{verified}, \mathsf{sid}, m, v'(m, \sigma))$ to V.

What can go wrong with second generation?

Alice wants a signature on what she received



$(\texttt{sign}, \mathsf{sid}, m)$

Functionality asks Adv for signing and verification algs

Adv can hard-code verify to always fail on strings of a specific form (e.g., fail if message contains Alice's ID)

Functionality A.2. $\mathcal{F}_{\text{sig-2nd}}$ (Example

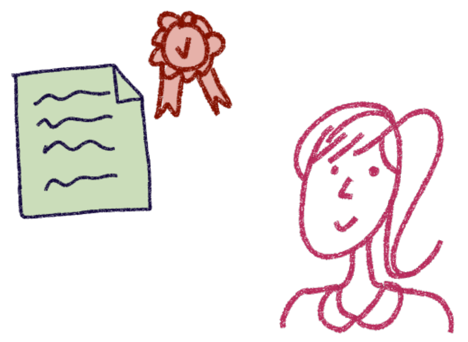
Key Generation. Upon receiving $(\texttt{keygen}, \mathsf{sid})$ from some party S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, hand $(\texttt{keygen}, \mathsf{sid})$ to the adversary. Upon receiving $(\texttt{algs}, \mathsf{sid}, s, v)$ from the adversary, where s is a description of a PPT ITM, and v is a description of a *deterministic* polytime ITM, output $(\texttt{VerificationAlgorithm}, \mathsf{sid}, v)$ to S.

Sign. Upon receiving $(\texttt{sign}, \mathsf{sid}, m)$ from S, let $\sigma = s(m)$, and verify that $v(m, \sigma) = 1$. If so, then output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to the caller P_i and record the entry (m, σ). Else, output an error message to S and halt.

Verify. On receiving the value $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ from some party V do: If $v' = v$, the signer is not corrupted, $v(m, \sigma) = 1$, and no entry (m, σ') for any σ' is recorded, then output an error message to S and halt. Else, output $(\texttt{verified}, \mathsf{sid}, m, v'(m, \sigma))$ to V.

What can go wrong with second generation?

Alice wants a signature on what she received



$(\texttt{sign}, \mathsf{sid}, m)$

ERROR!

Functionality asks Adv for signing and verification algs

Adv can hard-code verify to always fail on strings of a specific form (e.g., fail if message contains Alice's ID)

Functionality A.2. $\mathcal{F}_{\text{sig-2nd}}$ (Example

Key Generation. Upon receiving $(\texttt{keygen}, \mathsf{sid})$ from some party S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, hand $(\texttt{keygen}, \mathsf{sid})$ to the adversary. Upon receiving $(\texttt{algs}, \mathsf{sid}, s, v)$ from the adversary, where s is a description of a PPT ITM, and v is a description of a *deterministic* polytime ITM, output $(\texttt{VerificationAlgorithm}, \mathsf{sid}, v)$ to S.

Sign. Upon receiving $(\texttt{sign}, \mathsf{sid}, m)$ from S, let $\sigma = s(m)$, and verify that $v(m, \sigma) = 1$. If so, then output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to the caller P_i and record the entry (m, σ). Else, output an error message to S and halt.

Verify. On receiving the value $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ from some party V do: If $v' = v$, the signer is not corrupted, $v(m, \sigma) = 1$, and no entry (m, σ') for any σ' is recorded, then output an error message to S and halt. Else, output $(\texttt{verified}, \mathsf{sid}, m, v'(m, \sigma))$ to V.

Third generation: Pre-determined Signature Algorithms

- Specify the signature scheme (e.g., ECDSA) and internally run the algorithms of the scheme
  - Side-steps the challenges with adversarial algorithms
  - Used mostly for threshold signatures
- Does not try to capture all signature schemes

What do we want from a fourth generation?

- Be compatible with as many existing signature schemes as possible
- Functionality should always provide perfect signatures to honest parties regardless of the behavior of the adversary
  - Adversary should never be able to violate security or render functionality useless (even with negligible probability)

Fourth generation Pokemon
my friend told me is unstoppable

This Work

- Introduce a new ideal functionality for signatures that can be realized by EUF-CMA* signatures and can't be disabled by an adversary
    - Prove equivalence to EUF-CMA*
- Compatible with how signatures are treated in consensus literature
    - Give the first modular analysis of Dolev-Strong
- Generalize our functionality to threshold signatures

*with the extra requirement that they're "consistent"

Our Approach

- Use the “second generation” as a starting point (adversary supplies algs)
- If the supplied algorithms are bad, have a fallback to continue working
  - Instead of throwing an error, act in a way that maintains how signatures are “supposed to” act
- Consider how signatures are “supposed to act” as invariants
  - When used in a larger protocol, can reason about in terms of invariants

Disclaimer: We don’t try (or claim) to do everything

- Although we want to be general, we focus on core, minimal properties
- Forgo inessential but possibly desirable properties
  - No notion of who “owns” a public key
  - No way of transferring signing powers to another party

Fallback: "Random Mode"

- Functionality guarantees certain invariants you'd expect from signatures
    - If these are about to be violated, functionality disregards algorithms and randomly samples future keys and signatures
- When used with a EUF-CMA signature against a poly-time adversary random mode doesn't activate
- Unfortunately adds a layer of complexity to the functionality itself to maintain bookkeeping

(Standard) Signature invariants

- **Correctness**
- **Unforgeability**
- **Consistency**

(Standard) Signature invariants

- **Correctness** → Signature pairs issued by the functionality should always verify
- **Unforgeability**
- **Consistency**

(Standard) Signature invariants

- **Correctness**
- **Unforgeability**
- **Consistency**

Signature pairs issued by the functionality should always verify

Verify should never pass for messages that have never been signed

(Standard) Signature invariants

- **Correctness** → Signature pairs issued by the functionality should always verify
- **Unforgeability** → Verify should never pass for messages that have never been signed
- **Consistency** → Identical verification queries give the same response

(Standard) Signature invariants

- **Correctness**
- **Unforgeability**
- **Consistency**

Signature pairs issued by the functionality should always verify

Verify should never pass for messages that have never been signed

Identical verification queries give the same response

Do not issue a signature σ on message m if verification has previously failed on (m, σ) under the same public key

(Non-Standard) Signature Functionality Invariants

- Functionality should never emit the same signature in response to signing queries on two different messages under the same public key

- Functionality should never emit the same public key twice in response to honest key-generation queries

Starting point: second generation functionality

Functionality A.2. $\mathcal{F}_{\text{sig-2nd}}$ (Example Second-Generation Signature Functionality)

Key Generation. Upon receiving $(\texttt{keygen}, \mathsf{sid})$ from some party S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, hand $(\texttt{keygen}, \mathsf{sid})$ to the adversary. Upon receiving $(\texttt{algs}, \mathsf{sid}, s, v)$ from the adversary, where s is a description of a PPT ITM, and v is a description of a *deterministic* polytime ITM, output $(\texttt{VerificationAlgorithm}, \mathsf{sid}, v)$ to S.

Sign. Upon receiving $(\texttt{sign}, \mathsf{sid}, m)$ from S, let $\sigma = s(m)$, and verify that $v(m, \sigma) = 1$. If so, then output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to the caller P_i and record the entry (m, σ). Else, output an error message to S and halt.

Verify. On receiving the value $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ from some party V do: If $v' = v$, the signer is not corrupted, $v(m, \sigma) = 1$, and no entry (m, σ') for any σ' is recorded, then output an error message to S and halt. Else, output $(\texttt{verified}, \mathsf{sid}, m, v'(m, \sigma))$ to V.

Starting point: second generation functionality

Receive algorithms from the adversary

Functionality A.2. $\mathcal{F}_{\text{sig-2nd}}$ (Example Second-Generation Signature Functionality)

Key Generation. Upon receiving ($\texttt{keygen}, \mathsf{sid}$) from some party S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, hand ($\texttt{keygen}, \mathsf{sid}$) to the adversary. Upon receiving ($\texttt{algs}, \mathsf{sid}, s, v$) from the adversary, where s is a description of a PPT ITM, and v is a description of a *deterministic* polytime ITM, output ($\texttt{VerificationAlgorithm}, \mathsf{sid}, v$) to S.

Sign. Upon receiving ($\texttt{sign}, \mathsf{sid}, m$) from S, let $\sigma = s(m)$, and verify that $v(m, \sigma) = 1$. If so, then output ($\texttt{signature}, \mathsf{sid}, m, \sigma$) to the caller P_i and record the entry (m, σ). Else, output an error message to S and halt.

Verify. On receiving the value ($\texttt{verify}, \mathsf{sid}, m, \sigma, v'$) from some party V do: If $v' = v$, the signer is not corrupted, $v(m, \sigma) = 1$, and no entry (m, σ') for any σ' is recorded, then output an error message to S and halt. Else, output ($\texttt{verified}, \mathsf{sid}, m, v'(m, \sigma)$) to V.

Starting point: second generation functionality

Receive algorithms from the adversary

Functionality A.2. $\mathcal{F}_{\text{sig-2nd}}$ (Example Second-Generation Signature Functionality)

Key Generation. Upon receiving $(\texttt{keygen}, \mathsf{sid})$ from some party S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, hand $(\texttt{keygen}, \mathsf{sid})$ to the adversary. Upon receiving $(\texttt{algs}, \mathsf{sid}, s, v)$ from the adversary, where s is a description of a PPT ITM, and v is a description of a *deterministic* polytime ITM, output $(\texttt{VerificationAlgorithm}, \mathsf{sid}, v)$ to S.

Sign. Upon receiving $(\texttt{sign}, \mathsf{sid}, m)$ from S, let $\sigma = s(m)$, and verify that $v(m, \sigma) = 1$. If so, then output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to the caller P_i and record the entry (m, σ). Else, output an error message to S and halt.

Verify. On receiving the value $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ from some party V do: If $v' = v$, the signer is not corrupted, $v(m, \sigma) = 1$, and no entry (m, σ') for any σ' is recorded, then output an error message to S and halt. Else, output $(\texttt{verified}, \mathsf{sid}, m, v'(m, \sigma))$ to V.

Enforce correctness and unforgeability

Starting point: second generation functionality

Receive algorithms from the adversary

Functionality A.2. $\mathcal{F}_{\text{sig-2nd}}$ (Example Second-Generation Signature Functionality)

Key Generation. Upon receiving $(\texttt{keygen}, \mathsf{sid})$ from some party S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, hand $(\texttt{keygen}, \mathsf{sid})$ to the adversary. Upon receiving $(\texttt{algs}, \mathsf{sid}, s, v)$ from the adversary, where s is a description of a PPT ITM, and v is a description of a *deterministic* polytime ITM, output $(\texttt{VerificationAlgorithm}, \mathsf{sid}, v)$ to S.

Sign. Upon receiving $(\texttt{sign}, \mathsf{sid}, m)$ from S, let $\sigma = s(m)$, and verify that $v(m, \sigma) = 1$. If so, then output $(\texttt{signature}, \mathsf{sid}, m, \sigma)$ to the caller P_i and record the entry (m, σ). Else, output an error message to S and halt.

Verify. On receiving the value $(\texttt{verify}, \mathsf{sid}, m, \sigma, v')$ from some party V do: If $v' = v$, the signer is not corrupted, $v(m, \sigma) = 1$, and no entry (m, σ') for any σ' is recorded, then output an error message to S and halt. Else, output $(\texttt{verified}, \mathsf{sid}, m, v'(m, \sigma))$ to V.

Enforce correctness and unforgeability

Tasks:
- Handle asking adversary for algorithms
- Define new behavior on “error” to avoid halting

Our unstoppable signature functionality

Functionality 3.1. $\mathcal{F}_{\text{sig}}$ (An Unstoppable Signature Functionality)

This functionality interacts with an ideal adversary $\mathcal{S}$ and a number of real parties (all of them denoted P) that is not a-priori known. For simplicity of description, we assume this functionality has *per-session* memory. That is, all stored and recalled values are associated with the particular session ID sid of the query that generated them. Note that P may refer to a different party in every interaction.

Initialization.

1. Ignore any message from any party P that contains some session ID sid until *after* party P sends $(\texttt{init}, \mathsf{sid})$ to $\mathcal{F}_{\text{sig}}$.
2. Upon receiving $(\texttt{init}, \mathsf{sid})$ for the *first time* for some particular sid, send $(\texttt{init}, \mathsf{sid})$ to $\mathcal{S}$ and wait.
3. Upon receiving any second message that contains the session ID sid after the first $(\texttt{init}, \mathsf{sid})$ message (regardless of whether the same party transmitted the two messages):
   (a) If the message arrived from $\mathcal{S}$ and is of the form $(\texttt{algs}, \mathsf{sid}, \Sigma)$ where $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}) := \Sigma$ is the description of three probabilistic Turing machines, store $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ and $s := |\Sigma|$ in memory and set the flag $\mathsf{rmode} := 0$.
   (b) Otherwise, set the flag $\mathsf{rmode} := 1$.

   Regardless, set the integers $\ell_{\mathsf{pk}} := 1$ and $\ell_{\mathsf{sig}} := 1$, and initialize the set of assigned public keys $\mathcal{K} := \emptyset$ and the set of assigned signatures $\mathcal{Q} := \emptyset$. If $\mathsf{rmode} = 1$, process the second message for sid using the interfaces below.

Key Generation.

4. Upon receiving $(\texttt{keygen}, \mathsf{sid})$ from a party P,
   (a) If $\mathsf{rmode} = 0$, then sample a uniformly random bit-string r_k of appropriate length,[a] and compute $(\mathsf{sk}, \mathsf{pk}) := \mathsf{Gen}(r_k)$. If $\mathsf{pk} \in \mathcal{K}$ or Gen does not terminate in s computational steps, then switch to random mode by setting $\mathsf{rmode} := 1$ and following the instruction below for the case that $\mathsf{rmode} = 1$.
   (b) If $\mathsf{rmode} = 1$, then sample $\mathsf{pk} \leftarrow \{0,1\}^{\ell_{\mathsf{pk}}} \setminus \mathcal{K}$ uniformly and set $\mathsf{sk} := \bot$ and $r_k := \bot$.

   Regardless, update $\mathcal{K} := \mathcal{K} \cup \{\mathsf{pk}\}$ in memory and increment ℓ_{pk} until $\{0,1\}^{\ell_{\mathsf{pk}}} \setminus \mathcal{K} \neq \emptyset$. Store $(\texttt{key}, \mathsf{sid}, P, \mathsf{pk}, \mathsf{sk}, r_k)$ in memory and send $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ to the caller P.

Signing.

5. Upon receiving $(\texttt{sign}, \mathsf{sid}, \mathsf{pk}, m)$ from a party P, update $\mathcal{K} := \mathcal{K} \cup \{\mathsf{pk}\}$, and increment ℓ_{pk} until $\{0,1\}^{\ell_{\mathsf{pk}}} \setminus \mathcal{K} \neq \emptyset$. Check if a record of the form $(\texttt{key}, \mathsf{sid}, P, \mathsf{pk}, \mathsf{sk}, r_k)$ exists in memory for any $\mathsf{sk} \in \{0,1\}^* \cup \{\bot\}$ and any r_k. If not, return $\bot$ to P. Otherwise:
   (a) If $\mathsf{rmode} = 0$, then sample a uniformly random bit-string r_σ of appropriate length,[a] compute $\sigma := \mathsf{Sign}(\mathsf{sk}, m; r_\sigma)$ and check the following conditions:
   - $(\texttt{sig}, \mathsf{sid}, \mathsf{pk}, m', \sigma, r_\sigma)$ exists in memory such that $m \neq m'$.
   - $(\texttt{bad-sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ exists in memory.
   - Sign does not terminate in $(|m|+1) \cdot s$ computational steps.

   If any of the above conditions holds, then switch to random mode by setting $\mathsf{rmode} := 1$ and following the instruction below for the case that $\mathsf{rmode} = 1$.
   (b) If $\mathsf{rmode} = 1$, then sample $\sigma \leftarrow \{0,1\}^{\ell_{\mathsf{sig}}} \setminus \mathcal{Q}$ and set $r_\sigma := \bot$.

   Regardless, update $\mathcal{Q} := \mathcal{Q} \cup \{\sigma\}$ and increment ℓ_{sig} until $\{0,1\}^{\ell_{\mathsf{sig}}} \setminus \mathcal{Q} \neq \emptyset$. Store $(\texttt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, r_\sigma)$ in memory and return $(\texttt{signature}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ to the caller P.

Verification.

6. Upon receiving $(\texttt{verify}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ from some party P, update $\mathcal{K} := \mathcal{K} \cup \{\mathsf{pk}\}$, and increment ℓ_{pk} until $\{0,1\}^{\ell_{\mathsf{pk}}} \setminus \mathcal{K} \neq \emptyset$. Next, scan the memory for records of the form $(\texttt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, *)$ or $(\texttt{bad-sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$, for any σ, and for a record of the form $(\texttt{key}, \mathsf{sid}, P', \mathsf{pk}, *, *)$ for any P'.[b]
   (a) If the sig record exists, then set $b := 1$.
   (b) If there is no sig record, but there is a key record and P' is an honest party, then set $b := 0$.
   (c) If there is no sig record, but the bad-sig record exists, then set $b := 0$.
   (d) If Steps 6a through 6c do not apply, and $\mathsf{rmode} = 1$, then set $b := 0$.
   (e) If Steps 6a through 6c do not apply, and $\mathsf{rmode} = 0$, then set $b \leftarrow \mathsf{Verify}(\mathsf{pk}, m, \sigma)$. If Verify does not produce output before $(|m|+1) \cdot s$ computational steps have elapsed, then terminate its execution, set $b := 0$, and switch to random mode by setting $\mathsf{rmode} := 1$ in memory.

   If, after evaluating the above conditions, $b = 0$ but the record $(\texttt{bad-sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ is not stored in memory, then store it.

   If, after evaluating the above conditions, $b = 1$ but no record of the form $(\texttt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, *)$ exists in memory, then store $(\texttt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, \bot)$.

   Regardless, update $\mathcal{Q} := \mathcal{Q} \cup \{\sigma\}$ in memory and increment ℓ_{sig} until $\{0,1\}^{\ell_{\mathsf{sig}}} \setminus \mathcal{Q} \neq \emptyset$. Finally, return $(\texttt{verified}, \mathsf{sid}, \mathsf{pk}, m, \sigma, b)$ to P.

Corruption.

7. Upon receiving $(\texttt{corrupt}, \mathsf{sid}, P)$ from $\mathcal{S}$, search the memory for all records of the form $(\texttt{key}, \mathsf{sid}, P, \mathsf{pk}, \mathsf{sk}, r_k)$, and for each such record compute the set $\mathcal{C}_{\mathsf{pk}}$ of all (m, σ, r_σ) such that there exists a record of the form $(\texttt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, r_\sigma)$ in memory. Return $(\texttt{corrupt}, \mathsf{sid}, P, \mathcal{C})$ to $\mathcal{S}$, where $\mathcal{C}$ is a set containing $(\mathsf{pk}, \mathsf{sk}, r_k, \mathcal{C}_{\mathsf{pk}})$ for every $(\texttt{key}, \mathsf{sid}, P, \mathsf{pk}, \mathsf{sk}, r_k)$ that was found.

[a] We assume that the amount of randomness that Gen, Sign, and Verify need is part of their description.
[b] P' may or may not be the same as P.

Our unstoppable signature functionality

I do *not* expect you to read all of this!

We are going to walk through the main points

Functionality 3.1. $\mathcal{F}_{\mathsf{sig}}$ (An Unstoppable Signature Functionality)

This functionality interacts with an ideal adversary $\mathcal{S}$ and a number of real parties (all of them denoted P) that is not a-priori known. For simplicity of description, we assume this functionality has *per-session* memory. That is, all stored and recalled values are associated with the particular session ID sid of the query that generated them. Note that P may refer to a different party in every interaction.

Initialization.

1. Ignore any message from any party P that contains some session

signatures $\mathcal{Q} := \emptyset$. If $\mathsf{rmode} = 1$, process the second message for sid using the interfaces below.

Key Generation.

4. Upon receiving $(\mathtt{keygen}, \mathsf{sid})$ from a party P,
   (a) If $\mathsf{rmode} = 0$, then sample a uniformly random bit-string r_k of appropriate length,[a] and compute $(\mathsf{sk}, \mathsf{pk}) := \mathsf{Gen}(r_k)$. If $\mathsf{pk} \in \mathcal{K}$ or Gen does not terminate in s computational steps, then switch to random mode by setting $\mathsf{rmode} := 1$ and following the instruction below for the case that $\mathsf{rmode} = 1$.
   (b) If $\mathsf{rmode} = 1$, then sample $\mathsf{pk} \leftarrow \{0,1\}^{\ell_{\mathsf{pk}}} \setminus \mathcal{K}$ uniformly and set $\mathsf{sk} := \perp$ and $r_k := \perp$.

   Regardless, update $\mathcal{K} := \mathcal{K} \cup \{\mathsf{pk}\}$ in memory and increment ℓ_{pk} until $\{0,1\}^{\ell_{\mathsf{pk}}} \setminus \mathcal{K} \neq \emptyset$. Store $(\mathtt{key}, \mathsf{sid}, P, \mathsf{pk}, \mathsf{sk}, r_k)$ in memory and send $(\mathtt{public\text{-}key}, \mathsf{sid}, \mathsf{pk})$ to the caller P.

Signing.

5. Upon receiving $(\mathtt{sign}, \mathsf{sid}, \mathsf{pk}, m)$ from a party P, update $\mathcal{K} := \mathcal{K} \cup \{\mathsf{pk}\}$, and increment ℓ_{pk} until $\{0,1\}^{\ell_{\mathsf{pk}}} \setminus \mathcal{K} \neq \emptyset$. Check if a record of the form $(\mathtt{key}, \mathsf{sid}, P, \mathsf{pk}, \mathsf{sk}, r_k)$ exists in memory for any $\mathsf{sk} \in \{0,1\}^* \cup \{\perp\}$ and any r_k. If not, return $\perp$ to P. Otherwise:
   (a) If $\mathsf{rmode} = 0$, then sample a uniformly random bit-string r_σ of appropriate length,[a] compute $\sigma := \mathsf{Sign}(\mathsf{sk}, m; r_\sigma)$ and check the following conditions:
      - $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m', \sigma, r_\sigma)$ exists in memory such that $m \neq m'$.
      - $(\mathtt{bad\text{-}sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ exists in memory.
      - Sign does not terminate in $(|m|+1) \cdot s$ computational steps.

      If any of the above conditions holds, then switch to random mode by setting $\mathsf{rmode} := 1$ and following the instruction below for the case that $\mathsf{rmode} = 1$.
   (b) If $\mathsf{rmode} = 1$, then sample $\sigma \leftarrow \{0,1\}^{\ell_{\mathsf{sig}}} \setminus \mathcal{Q}$ and set $r_\sigma := \perp$.

   Regardless, update $\mathcal{Q} := \mathcal{Q} \cup \{\sigma\}$ and increment ℓ_{sig} until $\{0,1\}^{\ell_{\mathsf{sig}}} \setminus \mathcal{Q} \neq \emptyset$. Store $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, r_\sigma)$ in memory and return $(\mathtt{signature}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ to the caller P.

Verification.

6. Upon receiving $(\mathtt{verify}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ from some party P, update $\mathcal{K} := \mathcal{K} \cup \{\mathsf{pk}\}$, and increment ℓ_{pk} until $\{0,1\}^{\ell_{\mathsf{pk}}} \setminus \mathcal{K} \neq \emptyset$. Next,

   If, after evaluating the above conditions, $b = 0$ but the record $(\mathtt{bad\text{-}sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ is not stored in memory, then store it.

   If, after evaluating the above conditions, $b = 1$ but no record of the form $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, *)$ exists in memory, then store $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, \perp)$.

   Regardless, update $\mathcal{Q} := \mathcal{Q} \cup \{\sigma\}$ in memory and increment ℓ_{sig} until $\{0,1\}^{\ell_{\mathsf{sig}}} \setminus \mathcal{Q} \neq \emptyset$. Finally, return $(\mathtt{verified}, \mathsf{sid}, \mathsf{pk}, m, \sigma, b)$ to P.

Corruption.

7. Upon receiving $(\mathtt{corrupt}, \mathsf{sid}, P)$ from $\mathcal{S}$, search the memory for all records of the form $(\mathtt{key}, \mathsf{sid}, P, \mathsf{pk}, \mathsf{sk}, r_k)$, and for each such record compute the set $\mathcal{C}_{\mathsf{pk}}$ of all (m, σ, r_σ) such that there exists a record of the form $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, r_\sigma)$ in memory. Return $(\mathtt{corrupt}, \mathsf{sid}, P, \mathcal{C})$ to $\mathcal{S}$, where $\mathcal{C}$ is a set containing $(\mathsf{pk}, \mathsf{sk}, r_k, \mathcal{C}_{\mathsf{pk}})$ for every $(\mathtt{key}, \mathsf{sid}, P, \mathsf{pk}, \mathsf{sk}, r_k)$ that was found.

[a] We assume that the amount of randomness that Gen, Sign, and Verify need is part of their description.

[b] P' may or may not be the same as P.

Our unstoppable signature functionality

No guarantee the adversary will give algorithms in a timely fashion
Can either:

- Use library functionality of Canetti, Jain, Swanberg, Varia '22
- Initialization stage that has the parties activate the functionality without expecting a response

Functionality 3.1. $\mathcal{F}_{\mathsf{sig}}$ (An Unstoppable Signature Functionality)

This functionality interacts with an ideal adversary $\mathcal{S}$ and a number of real parties (all of them denoted P) that is not a-priori known. For simplicity of description, we assume this functionality has *per-session* memory. That is, all stored and recalled values are associated with the particular session ID sid of the query that generated them. Note that P may refer to a different party in every interaction.

Initialization.

1. Ignore any message from any party P that contains some session ID sid until *after* party P sends $(\mathtt{init}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{sig}}$.
2. Upon receiving $(\mathtt{init}, \mathsf{sid})$ for the *first time* for some particular sid, send $(\mathtt{init}, \mathsf{sid})$ to $\mathcal{S}$ and wait.
3. Upon receiving any second message that contains the session ID sid after the first $(\mathtt{init}, \mathsf{sid})$ message (regardless of whether the same party transmitted the two messages):
   (a) If the message arrived from $\mathcal{S}$ and is of the form $(\mathtt{algs}, \mathsf{sid}, \Sigma)$ where $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}) := \Sigma$ is the description of three probabilistic Turing machines, store $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ and $s := |\Sigma|$ in memory and set the flag $\mathsf{rmode} := 0$.
   (b) Otherwise, set the flag $\mathsf{rmode} := 1$.

   Regardless, set the integers $\ell_{\mathsf{pk}} := 1$ and $\ell_{\mathsf{sig}} := 1$, and initialize the set of assigned public keys $\mathcal{K} := \emptyset$ and the set of assigned signatures $\mathcal{Q} := \emptyset$. If $\mathsf{rmode} = 1$, process the second message for sid using the interfaces below.

Key Generation.

4. Upon receiving $(\mathtt{keygen}, \mathsf{sid})$ from a party P,
   (a) If $\mathsf{rmode} = 0$, then sample a uniformly random bit-string r_k of appropriate length,[a] and compute $(\mathsf{sk}, \mathsf{pk}) := \mathsf{Gen}(r_k)$. If $\mathsf{pk} \in \mathcal{K}$ or Gen does not terminate in s computational steps, then switch to random mode by setting $\mathsf{rmode} := 1$ and following the instruction below for the case that $\mathsf{rmode} = 1$.
   (b) If $\mathsf{rmode} = 1$, then sample $\mathsf{pk} \leftarrow \{0,1\}^{\ell_{\mathsf{pk}}} \setminus \mathcal{K}$ uniformly and set $\mathsf{sk} := \perp$ and $r_k := \perp$.

   Regardless, update $\mathcal{K} := \mathcal{K} \cup \{\mathsf{pk}\}$ in memory and increment ℓ_{pk} until $\{0,1\}^{\ell_{\mathsf{pk}}} \setminus \mathcal{K} \neq \emptyset$. Store $(\mathtt{key}, \mathsf{sid}, P, \mathsf{pk}, \mathsf{sk}, r_k)$ in memory and send $(\mathtt{public\text{-}key}, \mathsf{sid}, \mathsf{pk})$ to the caller P.

Signing.

5. Upon receiving $(\mathtt{sign}, \mathsf{sid}, \mathsf{pk}, m)$ from a party P, update $\mathcal{K} := \mathcal{K} \cup \{\mathsf{pk}\}$, and increment ℓ_{pk} until $\{0,1\}^{\ell_{\mathsf{pk}}} \setminus \mathcal{K} \neq \emptyset$. Check if a record of the form $(\mathtt{key}, \mathsf{sid}, P, \mathsf{pk}, \mathsf{sk}, r_k)$ exists in memory for any $\mathsf{sk} \in \{0,1\}^* \cup \{\perp\}$ and any r_k. If not, return $\perp$ to P. Otherwise:
   (a) If $\mathsf{rmode} = 0$, then sample a uniformly random bit-string r_σ of appropriate length,[a] compute $\sigma := \mathsf{Sign}(\mathsf{sk}, m; r_\sigma)$ and check the following conditions:
   - $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m', \sigma, r_\sigma)$ exists in memory such that $m \neq m'$.
   - $(\mathtt{bad\text{-}sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ exists in memory.
   - Sign does not terminate in $(|m|+1) \cdot s$ computational steps.

   If any of the above conditions holds, then switch to random mode by setting $\mathsf{rmode} := 1$ and following the instruction below for the case that $\mathsf{rmode} = 1$.
   (b) If $\mathsf{rmode} = 1$, then sample $\sigma \leftarrow \{0,1\}^{\ell_{\mathsf{sig}}} \setminus \mathcal{Q}$ and set $r_\sigma := \perp$.

   Regardless, update $\mathcal{Q} := \mathcal{Q} \cup \{\sigma\}$ and increment ℓ_{sig} until $\{0,1\}^{\ell_{\mathsf{sig}}} \setminus \mathcal{Q} \neq \emptyset$. Store $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, r_\sigma)$ in memory and return $(\mathtt{signature}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ to the caller P.

Verification.

6. Upon receiving $(\mathtt{verify}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ from some party P, update $\mathcal{K} := \mathcal{K} \cup \{\mathsf{pk}\}$, and increment ℓ_{pk} until $\{0,1\}^{\ell_{\mathsf{pk}}} \setminus \mathcal{K} \neq \emptyset$. Next, scan the memory for records of the form $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, *)$ or $(\mathtt{bad\text{-}sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$, for any σ, and for a record of the form $(\mathtt{key}, \mathsf{sid}, P', \mathsf{pk}, *, *)$ for any P'.[b]
   (a) If the $\mathtt{sig}$ record exists, then set $b := 1$.
   (b) If there is no $\mathtt{sig}$ record, but there is a $\mathtt{key}$ record and P' is an honest party, then set $b := 0$.
   (c) If there is no $\mathtt{sig}$ record, but the $\mathtt{bad\text{-}sig}$ record exists, then set $b := 0$.
   (d) If Steps 6a through 6c do not apply, and $\mathsf{rmode} = 1$, then set $b := 0$.
   (e) If Steps 6a through 6c do not apply, and $\mathsf{rmode} = 0$, then set $b \leftarrow \mathsf{Verify}(\mathsf{pk}, m, \sigma)$. If Verify does not produce output before $(|m|+1) \cdot s$ computational steps have elapsed, then terminate its execution, set $b := 0$, and switch to random mode by setting $\mathsf{rmode} := 1$ in memory.

   If, after evaluating the above conditions, $b = 0$ but the record $(\mathtt{bad\text{-}sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ is not stored in memory, then store it.

   If, after evaluating the above conditions, $b = 1$ but no record of the form $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, *)$ exists in memory, then store $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, \perp)$.

   Regardless, update $\mathcal{Q} := \mathcal{Q} \cup \{\sigma\}$ in memory and increment ℓ_{sig} until $\{0,1\}^{\ell_{\mathsf{sig}}} \setminus \mathcal{Q} \neq \emptyset$. Finally, return $(\mathtt{verified}, \mathsf{sid}, \mathsf{pk}, m, \sigma, b)$ to P.

Corruption.

7. Upon receiving $(\mathtt{corrupt}, \mathsf{sid}, P)$ from $\mathcal{S}$, search the memory for all records of the form $(\mathtt{key}, \mathsf{sid}, P, \mathsf{pk}, \mathsf{sk}, r_k)$, and for each such record compute the set $\mathcal{C}_{\mathsf{pk}}$ of all (m, σ, r_σ) such that there exists a record of the form $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, r_\sigma)$ in memory. Return $(\mathtt{corrupt}, \mathsf{sid}, P, \mathcal{C})$ to $\mathcal{S}$, where $\mathcal{C}$ is a set containing $(\mathsf{pk}, \mathsf{sk}, r_k, \mathcal{C}_{\mathsf{pk}})$ for every $(\mathtt{key}, \mathsf{sid}, P, \mathsf{pk}, \mathsf{sk}, r_k)$ that was found.

[a] We assume that the amount of randomness that Gen, Sign, and Verify need is part of their description.
[b] P' may or may not be the same as P.

Our unstoppable signature functionality

No guarantee the adversary will give algorithms in a timely fashion
Can either:
- Use library functionality of Canetti, Jain, Swanberg, Varia '22
- Initialization stage that has the parties activate the functionality without expecting a response

Checking the inputted algorithms are maintain the invariants

Switches to random mode if anything is wrong

Functionality 3.1. $\mathcal{F}_{\mathsf{sig}}$ (An Unstoppable Signature Functionality)

This functionality interacts with an ideal adversary $\mathcal{S}$ and a number of real parties (all of them denoted P) that is not a-priori known. For simplicity of description, we assume this functionality has *per-session* memory. That is, all stored and recalled values are associated with the particular session ID sid of the query that generated them. Note that P may refer to a different party in every interaction.

Initialization.

1. Ignore any message from any party P that contains some session ID sid until *after* party P sends $(\mathtt{init}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{sig}}$.
2. Upon receiving $(\mathtt{init}, \mathsf{sid})$ for the *first time* for some particular sid, send $(\mathtt{init}, \mathsf{sid})$ to $\mathcal{S}$ and wait.
3. Upon receiving any second message that contains the session ID sid after the first $(\mathtt{init}, \mathsf{sid})$ message (regardless of whether the same party transmitted the two messages):
   (a) If the message arrived from $\mathcal{S}$ and is of the form $(\mathtt{algs}, \mathsf{sid}, \Sigma)$ where $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}) := \Sigma$ is the description of three probabilistic Turing machines, store $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ and $s := |\Sigma|$ in memory and set the flag $\mathsf{rmode} := 0$.
   (b) Otherwise, set the flag $\mathsf{rmode} := 1$.

   Regardless, set the integers $\ell_{\mathsf{pk}} := 1$ and $\ell_{\mathsf{sig}} := 1$, and initialize the set of assigned public keys $\mathcal{K} := \emptyset$ and the set of assigned signatures $\mathcal{Q} := \emptyset$. If $\mathsf{rmode} = 1$, process the second message for sid using the interfaces below.

Key Generation.

4. Upon receiving $(\mathtt{keygen}, \mathsf{sid})$ from a party P:
   (a) If $\mathsf{rmode} = 0$, then sample a uniformly random bit-string r_k of appropriate length,[a] and compute $(\mathsf{sk}, \mathsf{pk}) := \mathsf{Gen}(r_k)$. If $\mathsf{pk} \in \mathcal{K}$ or Gen does not terminate in s computational steps, then switch to random mode by setting $\mathsf{rmode} := 1$ and following the instruction below for the case that $\mathsf{rmode} = 1$.
   (b) If $\mathsf{rmode} = 1$, then sample $\mathsf{pk} \leftarrow \{0,1\}^{\ell_{\mathsf{pk}}} \setminus \mathcal{K}$ uniformly and set $\mathsf{sk} := \bot$ and $r_k := \bot$.

   Regardless, update $\mathcal{K} := \mathcal{K} \cup \{\mathsf{pk}\}$ in memory and increment ℓ_{pk} until $\{0,1\}^{\ell_{\mathsf{pk}}} \setminus \mathcal{K} \neq \emptyset$. Store $(\mathtt{key}, \mathsf{sid}, P, \mathsf{pk}, \mathsf{sk}, r_k)$ in memory and send $(\mathtt{public\text{-}key}, \mathsf{sid}, \mathsf{pk})$ to the caller P.

Signing.

5. Upon receiving $(\mathtt{sign}, \mathsf{sid}, \mathsf{pk}, m)$ from a party P, update $\mathcal{K} := \mathcal{K} \cup \{\mathsf{pk}\}$, and increment ℓ_{pk} until $\{0,1\}^{\ell_{\mathsf{pk}}} \setminus \mathcal{K} \neq \emptyset$. Check if a record of the form $(\mathtt{key}, \mathsf{sid}, P, \mathsf{pk}, \mathsf{sk}, r_k)$ exists in memory for any $\mathsf{sk} \in \{0,1\}^* \cup \{\bot\}$ and any r_k. If not, return $\bot$ to P. Otherwise:
   (a) If $\mathsf{rmode} = 0$, then sample a uniformly random bit-string r_σ of appropriate length,[a] compute $\sigma := \mathsf{Sign}(\mathsf{sk}, m; r_\sigma)$ and check the following conditions:
   - $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m', \sigma, r_\sigma)$ exists in memory such that $m \neq m'$.
   - $(\mathtt{bad\text{-}sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ exists in memory.
   - Sign does not terminate in $(|m|+1) \cdot s$ computational steps.

   If any of the above conditions holds, then switch to random mode by setting $\mathsf{rmode} := 1$ and following the instruction below for the case that $\mathsf{rmode} = 1$.
   (b) If $\mathsf{rmode} = 1$, then sample $\sigma \leftarrow \{0,1\}^{\ell_{\mathsf{sig}}} \setminus \mathcal{Q}$ and set $r_\sigma := \bot$.

   Regardless, update $\mathcal{Q} := \mathcal{Q} \cup \{\sigma\}$ and increment ℓ_{sig} until $\{0,1\}^{\ell_{\mathsf{sig}}} \setminus \mathcal{Q} \neq \emptyset$. Store $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, r_\sigma)$ in memory and return $(\mathtt{signature}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ to the caller P.

Verification.

6. Upon receiving $(\mathtt{verify}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ from some party P, update $\mathcal{K} := \mathcal{K} \cup \{\mathsf{pk}\}$, and increment ℓ_{pk} until $\{0,1\}^{\ell_{\mathsf{pk}}} \setminus \mathcal{K} \neq \emptyset$. Next, scan the memory for records of the form $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, *)$ or $(\mathtt{bad\text{-}sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$, for any σ, and for a record of the form $(\mathtt{key}, \mathsf{sid}, P', \mathsf{pk}, *, *)$ for any P'.[b]
   (a) If the $\mathtt{sig}$ record exists, then set $b := 1$.
   (b) If there is no $\mathtt{sig}$ record, but there is a $\mathtt{key}$ record and P' is an honest party, then set $b := 0$.
   (c) If there is no $\mathtt{sig}$ record, but the $\mathtt{bad\text{-}sig}$ record exists, then set $b := 0$.
   (d) If Steps 6a through 6c do not apply, and $\mathsf{rmode} = 1$, then set $b := 0$.
   (e) If Steps 6a through 6c do not apply, and $\mathsf{rmode} = 0$, then set $b \leftarrow \mathsf{Verify}(\mathsf{pk}, m, \sigma)$. If Verify does not produce output before $(|m|+1) \cdot s$ computational steps have elapsed, then terminate its execution, set $b := 0$, and switch to random mode by setting $\mathsf{rmode} := 1$ in memory.

   If, after evaluating the above conditions, $b = 0$ but the record $(\mathtt{bad\text{-}sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ is not stored in memory, then store it.

   If, after evaluating the above conditions, $b = 1$ but no record of the form $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, *)$ exists in memory, then store $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, \bot)$.

   Regardless, update $\mathcal{Q} := \mathcal{Q} \cup \{\sigma\}$ in memory and increment ℓ_{sig} until $\{0,1\}^{\ell_{\mathsf{sig}}} \setminus \mathcal{Q} \neq \emptyset$. Finally, return $(\mathtt{verified}, \mathsf{sid}, \mathsf{pk}, m, \sigma, b)$ to P.

Corruption.

7. Upon receiving $(\mathtt{corrupt}, \mathsf{sid}, P)$ from $\mathcal{S}$, search the memory for all records of the form $(\mathtt{key}, \mathsf{sid}, P, \mathsf{pk}, \mathsf{sk}, r_k)$, and for each such record compute the set $\mathcal{C}_{\mathsf{pk}}$ of all (m, σ, r_σ) such that there exists a record of the form $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, r_\sigma)$ in memory. Return $(\mathtt{corrupt}, \mathsf{sid}, P, \mathcal{C})$ to $\mathcal{S}$, where $\mathcal{C}$ is a set containing $(\mathsf{pk}, \mathsf{sk}, r_k, \mathcal{C}_{\mathsf{pk}})$ for every $(\mathtt{key}, \mathsf{sid}, P, \mathsf{pk}, \mathsf{sk}, r_k)$ that was found.

[a] We assume that the amount of randomness that Gen, Sign, and Verify need is part of their description.
[b] P' may or may not be the same as P.

Our unstoppable signature functionality

No guarantee the adversary will give algorithms in a timely fashion
Can either:
- Use library functionality of Canetti, Jain, Swanberg, Varia ’22
- Initialization stage that has the parties activate the functionality without expecting a response

Checking the inputted algorithms are maintain the invariants

Switches to random mode if anything is wrong

Random mode and bookkeeping for random mode

Functionality 3.1. $\mathcal{F}_{\text{sig}}$ (An Unstoppable Signature Functionality)

This functionality interacts with an ideal adversary $\mathcal{S}$ and a number of real parties (all of them denoted P) that is not a-priori known. For simplicity of description, we assume this functionality has *per-session* memory. That is, all stored and recalled values are associated with the particular session ID sid of the query that generated them. Note that P may refer to a different party in every interaction.

Initialization.

1. Ignore any message from any party P that contains some session ID sid until *after* party P sends (init, sid) to $\mathcal{F}_{\text{sig}}$.
2. Upon receiving (init, sid) for the *first time* for some particular sid, send (init, sid) to $\mathcal{S}$ and wait.
3. Upon receiving any second message that contains the session ID sid after the first (init, sid) message (regardless of whether the same party transmitted the two messages):
   (a) If the message arrived from $\mathcal{S}$ and is of the form (algs, sid, Σ) where (Gen, Sign, Verify) $:= \Sigma$ is the description of three probabilistic Turing machines, store (Gen, Sign, Verify) and $s := |\Sigma|$ in memory and set the flag rmode $:= 0$.
   (b) Otherwise, set the flag rmode $:= 1$.

   Regardless, set the integers $\ell_{\text{pk}} := 1$ and $\ell_{\text{sig}} := 1$, and initialize the set of assigned public keys $\mathcal{K} := \emptyset$ and the set of assigned signatures $\mathcal{Q} := \emptyset$. If rmode $= 1$, process the second message for sid using the interfaces below.

Key Generation.

4. Upon receiving (keygen, sid) from a party P,
   (a) If rmode $= 0$, then sample a uniformly random bit-string r_k of appropriate length,[a] and compute (sk, pk) $:=$ Gen(r_k). If pk $\in \mathcal{K}$ or Gen does not terminate in s computational steps, then switch to random mode by setting rmode $:= 1$ and following the instruction below for the case that rmode $= 1$.
   (b) If rmode $= 1$, then sample pk $\leftarrow \{0,1\}^{\ell_{\text{pk}}} \setminus \mathcal{K}$ uniformly and set sk $:= \perp$ and $r_k := \perp$.

   Regardless, update $\mathcal{K} := \mathcal{K} \cup \{\text{pk}\}$ in memory and increment ℓ_{pk} until $\{0,1\}^{\ell_{\text{pk}}} \setminus \mathcal{K} \neq \emptyset$. Store (key, sid, P, pk, sk, r_k) in memory and send (public-key, sid, pk) to the caller P.

Signing.

5. Upon receiving (sign, sid, pk, m) from a party P, update $\mathcal{K} := \mathcal{K} \cup \{\text{pk}\}$, and increment ℓ_{pk} until $\{0,1\}^{\ell_{\text{pk}}} \setminus \mathcal{K} \neq \emptyset$. Check if a record of the form (key, sid, P, pk, sk, r_k) exists in memory for any sk $\in \{0,1\}^* \cup \{\perp\}$ and any r_k. If not, return $\perp$ to P. Otherwise:
   (a) If rmode $= 0$, then sample a uniformly random bit-string r_σ of appropriate length,[a] compute $\sigma :=$ Sign(sk, m; r_σ) and check the following conditions:
   - (sig, sid, pk, m', σ, r_σ) exists in memory such that $m \neq m'$.
   - (bad-sig, sid, pk, m, σ) exists in memory.
   - Sign does not terminate in $(|m|+1)\cdot s$ computational steps.

   If any of the above conditions holds, then switch to random mode by setting rmode $:= 1$ and following the instruction below for the case that rmode $= 1$.
   (b) If rmode $= 1$, then sample $\sigma \leftarrow \{0,1\}^{\ell_{\text{sig}}} \setminus \mathcal{Q}$ and set $r_\sigma := \perp$.

   Regardless, update $\mathcal{Q} := \mathcal{Q} \cup \{\sigma\}$ and increment ℓ_{sig} until $\{0,1\}^{\ell_{\text{sig}}} \setminus \mathcal{Q} \neq \emptyset$. Store (sig, sid, pk, m, σ, r_σ) in memory and return (signature, sid, pk, m, σ) to the caller P.

Verification.

6. Upon receiving (verify, sid, pk, m, σ) from some party P, update $\mathcal{K} := \mathcal{K} \cup \{\text{pk}\}$, and increment ℓ_{pk} until $\{0,1\}^{\ell_{\text{pk}}} \setminus \mathcal{K} \neq \emptyset$. Next, scan the memory for records of the form (sig, sid, pk, m, σ, $*$) or (bad-sig, sid, pk, m, σ), for any σ, and for a record of the form (key, sid, P', pk, $*$, $*$) for any P'.[b]
   (a) If the sig record exists, then set $b := 1$.
   (b) If there is no sig record, but there is a key record and P' is an honest party, then set $b := 0$.
   (c) If there is no sig record, but the bad-sig record exists, then set $b := 0$.
   (d) If Steps 6a through 6c do not apply, and rmode $= 1$, then set $b := 0$.
   (e) If Steps 6a through 6c do not apply, and rmode $= 0$, then set $b \leftarrow$ Verify(pk, m, σ). If Verify does not produce output before $(|m|+1)\cdot s$ computational steps have elapsed, then terminate its execution, set $b := 0$, and switch to random mode by setting rmode $:= 1$ in memory.

   If, after evaluating the above conditions, $b = 0$ but the record (bad-sig, sid, pk, m, σ) is not stored in memory, then store it.
   If, after evaluating the above conditions, $b = 1$ but no record of the form (sig, sid, pk, m, σ, $*$) exists in memory, then store (sig, sid, pk, m, σ, $\perp$).
   Regardless, update $\mathcal{Q} := \mathcal{Q} \cup \{\sigma\}$ in memory and increment ℓ_{sig} until $\{0,1\}^{\ell_{\text{sig}}} \setminus \mathcal{Q} \neq \emptyset$. Finally, return (verified, sid, pk, m, σ, b) to P.

Corruption.

7. Upon receiving (corrupt, sid, P) from $\mathcal{S}$, search the memory for all records of the form (key, sid, P, pk, sk, r_k), and for each such record compute the set $\mathcal{C}_{\text{pk}}$ of all (m, σ, r_σ) such that there exists a record of the form (sig, sid, pk, m, σ, r_σ) in memory. Return (corrupt, sid, P, $\mathcal{C}$) to $\mathcal{S}$, where $\mathcal{C}$ is a set containing (pk, sk, r_k, $\mathcal{C}_{\text{pk}}$) for every (key, sid, P, pk, sk, r_k) that was found.

[a] We assume that the amount of randomness that Gen, Sign, and Verify need is part of their description.
[b] P' may or may not be the same as P.

Plugging it into Dolev-Strong

- Requires care to handle other details (e.g., synchrony, parties learning the public keys of other parties)

- Resulting protocol is straightforward

  - Proof is in the PKI and signature hybrid model

Theorem 4.5. *The protocol* π_{DS} perfectly *UC-realizes* $\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}})$ *in the* $(\mathcal{F}_{\mathsf{sig}}, \mathcal{F}_{\mathsf{pki}})$*-hybrid model against a (possibly unbounded) malicious adversary that can adaptively corrupt any set of parties.*

Summary of Results

- Introduce a new ideal functionality for signatures that can be realized by EUF-CMA* signatures and can’t be disabled by an adversary
- Give the first modular analysis of Dolev-Strong broadcast
- Future Works:
    - Can we simplify it?
    - Adding in other desirable properties such as sharing keys, unique signatures

ePrint/2024/1807